DEEP NEURAL LANGUAGE MODEL FOR TEXT CLASSIFICATION BASED ON CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

Abdalraouf Hassan

Under the Supervision of Dr. Ausif Mahmood

DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

AND ENGINEERING

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT

CONNECTICUT

May, 2018

DEEP NEURAL LANGUAGE MODEL FOR TEXT CLASSIFICATION BASED ON CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

Abdalraouf Hassan

Under the Supervision of Dr. Ausif Mahmood

Approvals

Committee Members

Name	Signature	Date
Dr. Ausif Mahood	Ausif Mahmood	4-27-2018
Dr. Shakour Abuzneid	AB Abuzneid	04/27/2018
Dr. Miad Faezipour	Miad Faezipour	04,27,2018
Dr. Xingguo Xiong	Xingguo Xiong	04/27/2018
Dr. Minkyu Kim		5/1/2018

Ph.D. Program Coordinator

Dr. Khaled M. Elleithy		5/8/2018

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood	Ausif Mahmood	4-27-2018

Dean, School of Engineering

Dr. Tarek M. Sobh	Tarek Sobh	5/10/2018

DEEP NEURAL LANGUAGE MODEL FOR TEXT CLASSIFICATION BASED ON CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

ABSTRACT

The evolution of the social media and the e-commerce sites produces a massive amount of unstructured text data on the internet. Thus, there is a high demand to develop an intelligent model to process it and extract a useful information from it. Text classification plays an important task for many Natural Language Processing (NLP) applications such as, sentiment analysis, web search, spam filtering, and information retrieval, in which we need to assign single or multiple predefined categories to a sequence of text.

In Neural Network Language Models learning long-term dependencies with gradient descent is difficult due to the vanishing gradient problem. Recently researchers started to increase the depth of the network in order to overcome the limitations of the existing techniques. However, increasing the depth of the network means increasing the number of the parameters, which makes the network computationally expensive, and more prone to overfitting. Furthermore, NLP systems traditionally treat words as discrete atomic symbols; the model can leverage small amounts of information regarding the relationship between the individual symbols.

In recent years, deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been applied to language

modeling with comparative, remarkable results. CNNs are a noble approach to extract higher-level features invariant to local translation. However, this method requires the stacking of multiple convolutional layers in order to capture long-term dependencies because of the locality of the convolutional and pooling layers.

In this dissertation, we introduce a joint CNN-RNN framework to overcome the problems in the existing deep learning models. Briefly, we applied an unsupervised neural language model to train initial word embeddings that are further tuned by our deep learning network, then the pre-trained parameters of the network are used to initialize the model. At a final stage, the proposed framework combines former information with a set of feature maps learned by a convolutional layer with long-term dependencies learned via Long-Short-Term Memory (LSTM). Empirically, we show that our approach, with slight hyperparameter tuning and static vectors, achieves outstanding results on multiple sentiment analysis benchmarks. Our approach outperforms several existing approaches in term of accuracy; our results are also competitive with the state-of-the-art results on the Stanford Large Movie Review (IMDB) dataset, and the Stanford Sentiment Treebank (SSTb) dataset. Our approach has a significant role in reducing the number of parameters and constructing the convolutional layer followed by the recurrent layer with no pooling layers. Our results show that we were able to reduce the loss of detailed, local information and capture long-term dependencies with an efficient framework that has fewer parameters and a high level of performance.

ACKNOWLEDGEMENTS

My thanks are wholly devoted to God who has helped me all the way to complete this work successfully. I owe a debt of gratitude to my family for understanding and encouragement.

This dissertation would not have been possible without the support of many people, I would like to thank my advisor and role model Prof. Ausif Mahmood for providing me with the perfect balance of guidance and freedom and helped me see pros and cons of so many decisions, small and large. I admire your ability to see the nuances in everything.

I want to sincerely thank my committee members Dr. Shakour Abuzneid, Dr. Xingguo Xiong, Dr. Maid Faezipour, and Dr. Minkyu Kim for being my dissertation committee members and helping me with their valuable and helpful comments, also I would like to thank Dr. Khaled Elleithy for his guidance and support through the journey of my PhD.

I dedicate my dissertation to the memory of the soul of my late mother and to my father. I wouldn't be where I am today without the amazing support, encouragement and love from my parents. It's the passion for exploration and adventure combined with determination and hard work that I learned from you. Those values are what led me through my PhD and let me have fun in the process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INRTODUCTION

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics. The objective of NLP is for computer to process or understand natural language in order to perform tasks that are useful (e.g., making appointments, buying things, question answering).

Text classification is an essential, and plays an important role for many NLP applications, such as sentiment analysis, information retrieval, web search, ranking and spam filtering, in which we need to assign single or multiple predefined categories to sequence of text. The classic approach of text classification typically starts with feature extraction stage then is followed by a classifier stage. Perhaps one of the popular technique for feature extraction is to represent a sentence as TF-IDF, then train a linear classifier, (e.g., a logistic regression or SVM) [1, 2].

Deep Neural Networks (DNNs) based model has shown very good results for several tasks in NLP [3-12].Despite the good performance of these models, in practice they are relatively slow at training and testing time; which restrain them for the use of a large scale of data, and it requires to stack many neural network layers in order to capture long-term dependencies in sequence of texts.

Deep learning approaches for NLP start with an input sentence is denoted as a sequence of word, each word is presented as a one-hot vector, and then each word in the

sequence is projected into a continuous vector space by being multiplied with a weight matrix, forming a sequence of dense, and these sequence then fed into deep neural network, which processes the sequence in multiple layer. Resulting in a prediction probability. This whole network is tuned jointly to maximize the classification accuracy on a training set. However, one-hot-vector makes no assumption about the similarity of words, moreover it is very high dimensional [13, 14]. Most of the recent deep neural network approaches typically require the text input to be represented as a fixed-length vector.

Perhaps the most common fixed-length representation for texts is the bag-of-words or n-gram [15, 16], because of it is simplicity, efficiency and often surprising accuracy. However, the bag-of-words has many disadvantages such as: first it is ignore the semantics of words, second the word order is lost, and therefore different sentences can have exactly the same representation, as long as the same words are used. N-grams consider the word order in short context. However, n-grams models suffers from data sparsity and high dimensionality. Bag-of-words and n-grams models have very small sense about the semantics of the words, more formally the distance between the words.

Recently, models based on Neural Networks have become increasingly popular [1, 4, 5, 9, 12-14, 17-30]; it has become possible to train more complex models on much larger dataset. They typically outperform the simple models. Perhaps the most efficient model is to use distributed representation of word [31, 32]. For instance neural network language models outperform n-gram models [13, 20, 33-35]. Currently for text classification problems a linear classifiers are considered to be the conventional approach and strong Baselines [1, 15, 36]. Regardless of their simplicity, the linear

classifier often obtains the state-of-the-art performances especially when the correct features are selected.

Deep neural network methods jointly implement feature extraction and classification for document classification [3, 4, 8, 17, 37]. The deep neural network based approach convention, in most cases, is an input document represented as a sequence of words, and each sequence is then represented as one-hot-vector, each word in the sequence is projected into a continuous vector space by multiplying it with weight matrix, forming a sequence of dense, real valued vector. This sequence is then fed into a deep neural network, which processes the sequence in multiple layers, finally resulting in prediction probability. This pipeline is tuned jointly to maximize the classification accuracy on training set [8, 11, 12, 17, 29, 30, 38-41].

Convolutional Neural Network (CNN) has recently accomplished a remarkable performance on the essentially significant task of sentence classification [17, 42, 43]. However, these models require professionals to specify an exact model architecture and set accompanying hyper-parameters, including the filter region size.

Recent work by [30] consists of multi layers of CNN and max pooling, similar to the architecture proposed by [41] in computer vision. In the first stage, each layer will extract features from small overlapping windows of the input sequence and pools over small non-overlapping windows by taking the maximum activation in the window. This is applied recursively for many times. The final convolutional layers are then flattened to form a vector, which feeds into a small number of fully connected layers followed by a classification layer.

We observed that that network requires many convolutional and pooling layers

in order to capture long-term dependencies, because of the locality of the convolutional and pooling. As the length of the input grows, this become crucial; that was the motivation behind [30] to investigate deep convolutional network to overcome these issues. [29] Investigated the combination of neural network architecture of CNN and Recurrent Neural Network (RNN) in order to encode character input, which was implemented to learn high-level feature input sequences of character level to capture sub word information. However, this model performs better only when a large number of classes are available. Another successful model applied RNN for NLP was introduced by [18, 44]; it confirmed that RNN is able to capture long-term dependencies even in the case of a single layer.

Today RNN is the lead approach for many NLP applications. Recursive Neural Network was applied to sentence classification [45]; configuration function is defined in this model and recursively applied at each node of the parse tree of an input sentence. In order to extract a feature vector of the sentence, the model relies on an external parser.

NLP is a vast area of computer science that is concerned with the interaction between computers and human language. Language modeling is a fundamental task in artificial intelligence and NLP. A language model is formalized as a probability distribution over a sequence of words. Recently, deep learning models have achieved remarkable results in speech recognition [22] and computer vision [41].

Text classification plays an important role in many NLP applications, such as sentiment analysis, spam filtering, email categorization, information retrieval, web search, ranking , and document classification [46, 47], in which one needs to assign predefined categories to a sequence of text. A popular and common method to represent

texts is bag-of-words. However, the bag-of-words method loses the words order and ignores the semantics of words. N-gram models are popular for statistical language modeling and usually perform the best [1, 34, 35]. However, an n-gram model suffers from data sparsity [13].

Neural Networks have become increasingly popular [13, 20, 26]; it has become possible to train more complex models on a much larger dataset. They outperform n-gram models and overcome the data sparsity problem [13]; semantically similar words are close in vector space. The embedding of rare words is poorly estimated, which leads to higher perplexities for rare words. With the progress of machine learning in recent years, it has become possible to train more complex models on much larger data sets [14, 17, 22, 26, 41]. The distributed representation of words is one of the most successful concepts, and it helps learning algorithms achieve better performance [26].

Convolutional Neural Networks (CNN) [48] recently achieved very successful results in computer vision [41]. A CNN considers feature extraction and classification as one joint task. This idea has been improved by stacking multiple convolutional and pooling layers, which sequentially extract a hierarchical representation of the input [6, 43, 48].

We investigate Recurrent Neural Networks (RNNs) as an alternative for pooling layers in deep neural network language models to perform a sentiment analysis of a short text. Most of the deep learning architectures for NLP require stacking many layers to capture long-term dependences due to the locality of the convolutional and pooling layers [30]. Our architecture was inspired by the recent success of RNNs in NLP applications and the fact that RNNs can capture long-term dependencies even with one

single layer [44]. We were also inspired by the successful work proposed in [17], where a single layer of CNN was applied for sentence classification.

It turns out that providing the network with good initialization parameters can have a significant impact on the accuracy of the trained model and capturing the long-term dependencies more efficiently. In this paper, we present a joint CNN and RNN architecture that takes the local features extracted by a CNN as the input for an RNN for a sentiment analysis of short texts. We propose a new framework that exploits and combines convolutional and recurrent layers into one single model on top of pre-trained word vectors. We utilize long short-term memory (LSTM) as a substitute for pooling layers in order to reduce the loss of detailed, local information and capture long-term dependencies across the input sequence.

In this dissertation, I propose a neural language model that leverages both convolutional and recurrent layers to efficiently perform text classification tasks. Based on our observation from the work proposed in [3, 6, 30, 43] CNN architecture must have many layers to capture long-term dependencies in an input sentence. Our work is also inspired from the fact that recurrent layers are able to capture long-term dependences with one single layer [44]. In our model, we utilize a recurrent layer LSTM as substitutes for the pooling layer in order to reduce the loss of detailed local information and capture long-term dependencies. Surprisingly, our model achieved comparable results on two sentiment analysis benchmarks with less number of parameters. We show that it is possibly to use a much smaller model to achieve the same level of classification performance when recurrent layer combined with convolutional layer.

1.1 Research Problem and Motivation

Sentiment analysis for short texts becomes a challenge because of the limit amount of the contextual information they usually contain. Furthermore, learning good vector representations for sentences is a challenging task and an ongoing research area. Moreover, learning long-term dependencies with gradient descent is difficult in neural network language model because of the vanishing gradients problem, and natural language processing systems traditionally treat words as discrete atomic symbols; the model can leverage small amounts of information regarding the relationship between the individual symbols.

Recently, it became more common to use a deep neural network for natural language processing systems. Within NLP much of the work with deep leering method has involved learning word vector representation through neural language models and performing composition over the learned word vectors for classification.

In the input sequence it is possible for the gap between the relevant information to become very large, and this becomes more complex as the length of the input sequence grows. Furthermore, we observed that most of the existing deep learning models for NLP has increased the depth of the network in order to capture long-term dependencies. However increasing the depth of the network lead to increasing the number of the parameters. Moreover, it will causes the problem of vanishing gradients, the network will be more prone to overfitting and difficult to optimize.

Convolutional Neural Network (CNN) recently achieved a remarkable results in

NLP systems. The objective of using the convolutional layer in the CNN based model is for it to learn to extract higher-level features that are invariant to local translation, and, by assembling multiple convolutional layers, the model can extract higher-level translation invariant features from the input sequence. Regardless of this advantage, we observed that most of the existing deep models require multiple layers of convolutional to capture long-term dependencies, and that is because of the locality of the convolutional and pooling layers. This issue becomes more crucial as the length of the input sequence grows.

Contrary to the convolutional layer, in Recurrent Neural Network (RNN) based model the recurrent layer is able to capture long-term dependencies even when there is only a single layer, because of the hidden state is computed in the whole input sequence. However, the recurrent layer is computationally more expensive, the computational complexity grows linearly with respect to the input sequence, and most of the computations need to be done sequentially, this in contrast to the convolutional layer for which computations can be efficiently done in parallel.

Based on these observation, in this dissertation I proposed a deep neural network language model that can capture long-term dependencies in the document more efficiently for the task of classification. Most of the combination CNN-RNN models applied several types of pooling. We argue that the pooling layer is the reason for lost details in local information, because the pooling layer only captures the most important feature in a sentence; therefore, we exclude the pooling layer and utilized a recurrent layer to capture long-term dependencies more efficiently and reduce the number of the parameters in the architecture.

1.2 Contributions

In this dissertation we proposed a novel deep neural network language model that employs a Convolutional Neural Network (CNN) and Recurrent Neural Network Long Short-Term Memory (RNN-LSTM) over pre-trained word vectors word2vec to perform text classification tasks. We exploit LSTM as a substitute of pooling layer in CNN to reduce the loss of detailed local information and capture long term dependencies in sequence of sentences.

We introduced an architecture that focuses on parameter reduction in the network, while also capturing long-term dependencies more efficiently in terms of accuracy. Our contributions are as follow; In order to capture semantics and syntactic of subword information. Word embeddings are initialized with unsupervised pre-trained word vectors [14, 26], which is trained on a large unsupervised collection of words. We constructed a customized single convolutional layer with multiple filter size to extract higher-level features from the pre-trained word vectors.

We employed a CNN to further refine the embeddings on a distance-supervised dataset, word embedding served as the input to our model in which windows of different length and various weight matrices are applied to generate a number of feature maps. The word embeddings and other parameters of the network obtained at the previous stage are used to initialize the same framework. After the convolutional operations, we removed the pooling layer and take the encoded feature maps as an input to Long Sort-Tern Memory Recurrent Neural Network (LSTM-RNN) to asset the proposed model to capture long-term dependencies.

The deep learning framework takes advantage of the encoded local features extracted from the CNN model and the long-term dependencies captured by the RNN model. Our results demonstrated that the proposed framework achieves competitive performance with fewer parameters. The proposed model is simple and efficient with significantly fewer parameters, which means less memory consumption. The proposed model is more compact, and less prone to overfitting.

CHAPTER 2: LITERATURE SURVEY

Text classification is a classic topic in Natural Language Processing (NLP), in which one needs to assign predefined categories to free-text documents. The range of text classification research goes from designing the best features to choosing the best possible machine learning classifiers. To date, almost all techniques of text classification are based on words, in which simple statics of some ordered word combinations (such as n-grams) usually perform the best [1].

The goal of NLP is to process text with computers in order to analyze it, to extract information and eventually to represent the same information differently. We may want to associate categories to part of the text (e.g. POS tagging or sentiment analysis). Structure text differently (e.g. parsing), or convert it to some other form which preserves all or part of the content (e.g. machine translation, summarization).

2.1 Traditional Methods for Natural Language Processing

Text classification is significant for NLP systems, where there has been an enormous amount of research on sentence classification tasks, specifically on sentiment analysis. NLP systems classically treat words as discrete, atomic symbols where the model leverages a small amount of information regarding the relationship between the individual symbols.

A simple and efficient baseline method for a sentence structure is to represent the sentence as a bag-of-words and then train a linear classifier (e.g., a logistic regression). However, the bag-of-words approach omits all of the information about the semantics and ordering of words [4, 49]. N-gram models are another popular method to represent a sentence. This method usually performs the best [1].

Words are projected to a high-dimensional space, and then the embedding is combined to obtain a fixed-size representation of the input sentence, which later is used as an input to the classifier. Despite the fact that n-gram models take into account word ordering in short sentences, they do still suffer from data sparsity.

Overall, all simple techniques have limitations for certain tasks. Furthermore, linear classifiers do not share parameters among features and classes that might limit their generalization in the context of a large output, where some classes have few examples.

A popular solution for this problem is to use multilayer neural networks [4, 30], or to factorize the linear classifier into low-rank matrices [14, 36].

Text classification is an important task in Natural Language Processing (NLP), there is an enormous research activities on sentence classification tasks specifically on sentiment analysis.

NLP systems classically treat words as discrete atomic symbols; the model leverage small amount of information regarding the relationship between the individual symbols.

Simple and efficient baseline for sentence is to represent sentence as (bag-of-words), then train a linear classifier (e.g., a logistic regression). However, bag-of-words

omitted all the information's about words such as semantics and words ordering [1, 13].

2.1.1 N-gram Models

N-grams models is another popular method to represent sentence, it is usually perform the best [1], word are projected to high dimensional space, and then the embedding combined to acquire a fixed-size representation of input sentence, which later use as input to the classifier.

However, n-grams take in account the word ordering in short sentence, but it suffers from data sparsity and high dimensionality [13]. The probability of a sequence of symbols (usually words) is computed using a chain rule as

$$p(w) = \sum_{i=1}^{N} p(w_i | w_1 \dots w_{i-1}) \quad (2.1)$$

The model frequently used language models are based on the n-gram statistics, which are basically word co-occurrence frequencies. The maximum likelihood estimate of probability of word A in context H is then compute as

$$P(A|H) = \frac{C(HA)}{C(H)} \quad (2.2)$$

Where $C(HA)$ is the number of times that the $H\,A$ sequence of words has occurred in the training data? The context H can consist of several words, for the usual trigram models $|H| = 12$ for $H = \theta$, the model is called unigram, and it does not take into account history. As many of these probability estimates are going to be zero (for all words that were not seen in the training data in a particular context H), smoothing needs to be applied.

This works by redistributing probabilities between seen and unseen (zero-

frequency) events, by exploiting the fact that some estimates, mostly those based on single observations, are greatly over-estimated detailed overview of common smoothing techniques and empirical evaluation can be found in [50].

Simple techniques usually has limitation for some certain tasks. Furthermore, linear classifiers do not share parameters among features and class, perhaps that might limits their generalization in the context of large output where some classes have few examples.

Popular solution for this problem are to use multilayer neural networks or to factorize the linear classifier into low rank matrices [14, 30].

2.1.2 Structured Language Models

The statistical language modeling was criticized greatly by the linguists from the first days of its existence. There are many exampled showing that words in a sentence are often related, even if they do not lie next to each other.

It can be shown that such patterns cannot be effectively encoded using a finite state machine (n-gram models belong to this family of computational models). However, these pattern can be often effectively described while using for example context free grammars.

This was the motivation for the structured language models that attempt to bridge differences between the linguistic theories and the statistical models of the natural languages.

The sentence is viewed as a tree structure generated by a context free grammar, where leafs are individual words and nodes are non-terminal symbols. The statistical

approach is employed when constructing the tree: the derivations have assigned probabilities that are estimated from the training data, thus every new sentence can be assigned probability of being generated by the given grammar.

The advantage of these models is in their theoretical ability to represent patters in a sentence across many words. However there are many disadvantages of the structures language models: computational complexity and sometimes unstable behavior, ambiguity, questionable performance when applied to spontaneous speech, large amount of manual work that has to be done by expert linguists is often required.

2.1.3 Word Vector Representations

The majority of rule-based and statistical language processing algorithms regard word as atomic symbols. This translates to a very sparse vector representation of the size of the vocabulary and with a single 1 at the index location of the current word.

This so called “one-hot” representations has the problem that it does not capture any type of similarity between two words. So if a model sees “cat” in a surrounding context it cannot use this information when it sees “dog” at the same location during test time.

[4] induced a model to compute an embedding, the idea is to construct a neural network that outputs high scores for windows that occur in a large unlabeled corpus and low scores for windows where one word is replaced by a random word.

When such a network is optimized via gradient ascent the derivatives backpropagated into a word embedding matrix $L \in \mathbb{R}^{n \times V}$, where V is the size of the vocabulary.

[3, 5] adapted neural network architecture for training a language task. By leveraging a large amount of unlabeled text data, and induced word embedding which were shown to boost generalization performance on all tasks.

[51] Proposed a related language model approach inspired from Restricted Boltzmann Machines. However, word representations are perhaps more commonly inferred from n-gram language modelling rather than smoothed language models.

One popular approach is the Brown clustering algorithm [52] which builds hierarchical word clusters by maximizing the bigram's mutual information.

The induced word representation has been used with success in a wide variety of natural language processing tasks, including part of speech (POS) [53], Name Entity Recognition (NER) [54, 55], or parsing [56]. Other related approaches exist, like phrase clustering [57] which has been shown to work well for NER.

Finally, [58] Have recently proposed a smoothed language modelling approach based on a Hidden Markov Model, with success on POS and Chunking tasks.

2.2 Neural Networks: Basics and Definitions

In this section I will give a basic introduction to neural networks. Fig 2.1 show a single neurons which consists of input, an activation function and the output. Let the inputs be some n-dimensional vector $x \in \mathbb{R}^n$. The output is computed by the following function:

$$a = f(w^T x + b), \tag{2.3}$$

Where f defines the activation function. This function is also called a nonlinearity and commonly used examples are the sigmoid function:

$$f(x)sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{2.4}$$

Or the hyperbolic tangent function:

$$f(x) = tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}, \tag{2.5}$$

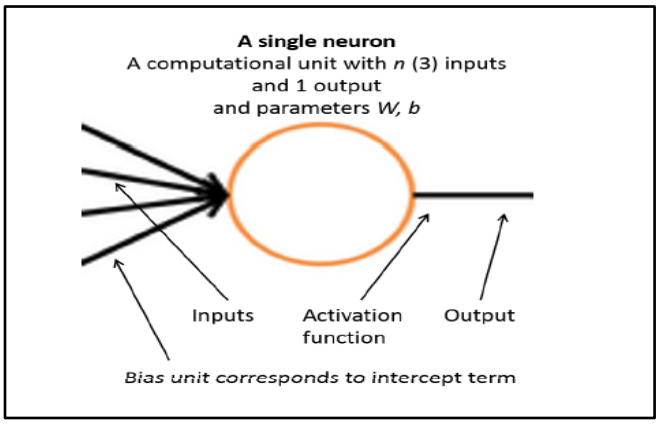

Figure 2.1. Definition of a single neuron with inputs, activation function and outputs [59].

The sigmoid activation function maps any real member to the [0, 1] interval. With this unit, the activation can be interpreted as the probability for the "neural unit" parameterized by w and the bias b to be on. Despite the loss of a probabilistic interpretation, the tanh function is often preferred in practice due to better empirical performance. Both nonlinearities are shown in Fig.2.2.

Various other recent nonlinearities exist such as the hard tanh or rectified linear: $f(x) = \max(0, x)$, which does not suffer from the vanishing gradient problem. The choice of nonlinearity should largely by selected by cross-validation over a development set.

While the clustering algorithms used for constructing class based language models are quite specific for the language modeling field, artificial neural networks can

be successful used for dimensionality reduction as well as for clustering, while being a very general machine learning technique. Therefore, it is a bit surprising that neural network based language models have gained attention only after [60], and not much earlier. Although a lot of interesting work on language modeling using neural networks was done much earlier [61].

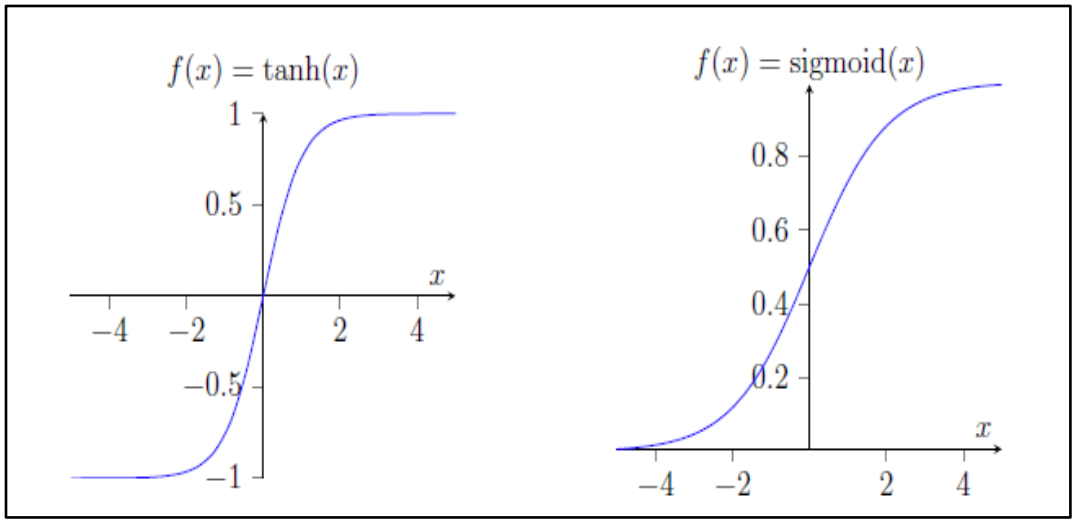

Figure 2.2. These are two commonly used nonlinearities [59].

Although it has been very surprising the NNLMs while very general and simple, have outperformed many of the competing techniques, including those that were developed specifically for modeling the language.

Neural network language models today are among the-state-of-the-art techniques. The main advantage of NNLMs over n-grams is that history is no longer seen as exact sequence of $n-1$ words H, but rather as a project of H into some lower dimensional space.

This reduces number of parameters in the model that have to be trained, resulting in automatic clustering of similar history.

This might sound the same as the motivation for class based models, the main difference is that NNLMs project all words into the same low dimensional space, and there can be many degrees of similarity between words.

The disadvantage of this models is very large computational complexity, which usually prohibits to train these models on full training set, using the full vocabulary.

2.2.1 Neural Network Language Models (NNLMs)

The use of artificial neural networks for sequence prediction is as the neural network techniques themselves. One of the first widely known attempts to describe language using neural networks was performed by [61], who applied recurrent neural network for modeling sentences of words generated by an artificial grammar.

The first serious attempt to build a statistical neural network based language model of real natural language, together with an empirical comparison of performance to standard techniques, n-gram models and class based models was done by [20], followed by [35], who did show that Neural Network Language Models (NNLMs) work very well in a state of the art in speech recognition systems, and are complementary to standard n-gram models [62].

However, no techniques or modifications of the original model that would significantly improve the ability of the model to capture patterns in the language were published.

[63, 64] investigated the integration of traditional features into the NNLM framework such as, part of speech tags or morphology information, the accuracy of the NNLM still the same until [65, 66] recently have shown that recurrent neural network

architecture can work actually better than the feedforward one.

Most of the research work did focus on overcoming practical problems when using these attractive models: the computational complexity was originally too high for real world tasks.

It was reported by [60] that training of the original neural net language model took almost a week using 40 CPUs for just single training epoch, and 10 to 20 epochs were needed for reaching optimal results.

Despite the limitations, the model provide 20% reduction of perplexity over a baseline n-gram model, after 5 training epochs.

Clearly, better results could have been expected if the computational complexity was not so restrictive, and most of the future research focused on this topic. The author proposed parallel training parallel training of the model on several CPUs, which was later repeated and extended by [62].

A very successful extension reduced computation between the hidden layer and the output layer in the model, using a trick that was originally proposed by Joshua Goodman for speeding up maximum entropy models [67].

2.2.2 Feedforward Neural Network Based Language Models (FFNNLMs)

The model was proposed by [20] start with input of the n-gram NNLM is formed by using a fixed length history of $n-1$ words, where each of the previous $n-1$ words is encoded using 1-of-V coding, where V is size of the vocabulary.

Thus, every word from the vocabulary is associated with a vector with length V,

where only one value corresponding to the index of given word in the vocabulary is 1 and all other values are 0.

This 1-of-V orthogonal representation of words is projected linearly to a lower dimensional space, using a shared matrix P, called also a projection matrix.

The matrix P is shared among words at different positions in the history, thus the matrix is the same when projecting word w_{t-1}, w_{t-2} etc. after the projection layer, a hidden layer with non-linear activation function (usually hyperbolic tangent or a logistic sigmoid) is used, with a dimensional of 100-300.

An output layer follows, with the size equal to the size of full vocabulary. After the network is trained, the output layer of n-gram NNLM represents probability distribution $P(w_t|w_{t-4}, w_{t-3}, w_{t-2}, w_{t-1})$.

[68] Proposed an alternative feedforward architecture of the NNLM. The problem of learning n-gram NNLM is decomposed into two steps: learning a bigram NNLM with only the previous word from the history encoded in the input layer. And then training an n-gram NNLM that projects words from the n-gram history into the lower dimensional space by using the already trained bigram NNLM.

Both models are simple feedforward neural networks with one hidden layer, thus this solution is simpler for implementation and for understanding than the original models [13, 20, 60]. It provides almost identical results as the original model.

2.3 Deep Learning Background

Most of current machine learning methods work well because of human-designed representations and inputs features. However, handcrafting feature is time-

consuming and features are often both over-specified an incomplete. If machine learning could learn features automatically, the entire process could be automated more efficiently and many task could be solved [3, 5, 21, 22, 41, 69].

When machine learning is applied only to the input features it relies on the optimizing weights to make the best final prediction.

Deep learning can be seen as putting back together representation learning with machine learning. It attempts to jointly learn good features, across multiple levels of increasing complexity and abstraction, and the final prediction.

2.4 Deep Learning for Natural Language Processing

After couple of pioneer works [3-5, 13, 60], the use of neural networks for NLP applications is attracting huge interest in the research community and they are systematically applied to all NLP tasks. However, while the use of deep neural networks in NLP has shown very good results for many tasks, it seems that they have not reached the level to outperform the state-of-the-art by a large margin. As it was observed in computer vision and speech recognition.

Deep learning can be perceived as putting back together representation learning with machine learning. It efforts to jointly learn good features, across multiple levels of increasing complexity and abstraction, and the final prediction.

Deep Learning achieved significant results in computer vision [41] and speech recognition [21, 22]. It has become more common to use deep neural methods in NLP applications; much of the work has involved learning word vector representations through neural language models, then performing composition over the learned word

vectors for classification [3, 6, 8, 17, 20, 38, 70-77].

Deep Neural Networks (DNNs) and representation learning approaches have led to new methods for solving the data sparsity problem. Several neural network based models for learning word representations followed this approach.

Word embedding is the neural representation of a word and is a real vector [13, 26]. Word embedding allows us to measure similarity between words by simply using the distance between two embedded vectors [14, 26].

Recently, researchers observed that is not necessary for deep neural network to perform at word level [29, 30].

As long as the document represented as one-hot-vector, the model could work without any change, regardless if each one-hot vector corresponds to a word [6, 30]. Character sequence proposed as an alternative to the one-hot vector [29].

Similar ideas also applied to dependency parsing in [28]. Deep Convolution Neural Network for NLP by [4] composed numerous of layers of convolutional and max pooling, it is identical to the convolutional architecture in the computer vision.

Several neural sentence models have been described. A general class of basic sentence models is that of Neural Bag-of-Words (NBoW) models. These generally consist of a projection layer that maps words, sub-word units or n-grams to high dimensional embeddings; the latter are then combined component-wise with an operation such as summation. The resulting combined vector is classified through one or more fully connected layers.

A model that adopts a more general structure provided by an external parse tree is the Recursive Neural Network (RecNN) [78-81]. At every node in the tree the

contexts at the left and right children of the node are combined by a classical layer. The weights of the layer are shared across all nodes in the tree. The layer computed at the top node gives a representation for the sentence.

The Recurrent Neural Network (RNN) is a special case of the recursive network where the structure that is followed is a simple linear chain [66, 82]. The RNN is primarily used as a language model, but may also be viewed as a sentence model with a linear structure.

The layer computed at the last word represents the sentence. Finally, a further class of neural sentence models is based on the convolution operation and the TDNN architecture [4, 83]. Certain concepts used in these models are central to the DCNN and we describe them next.

2.4.1 Windows-Based Neural Networks

DNNs have achieved significant results in computer vision [5] and speech recognition [21, 22]. Recently, it has become more common to use DNNs in NLP applications, where much of the work involves learning word representations through neural language models and then performing a composition over the learned word vectors for classification [5, 6, 8, 17, 21, 28, 40, 45, 76, 80].

These approaches have led to new methods for solving the data sparsity problem. Consequently, several neural network-based methods for learning word representations followed these approaches [71, 73-77].

DNNs jointly implement feature extraction and classification for text classification [5, 6, 23, 36]. DNN-based approaches usually start with an input text,

represented as a sequence of words, where each sequence is represented as one-hot vector; then, each word in the sequence is projected into a continuous vector space.

This happens by multiplying it with a weight matrix, which leads to the creation of a sequence of dense, actual, valued vectors [18, 84].

This sequence then feeds into a DNN, which processes the sequence in multiple layers, resulting in prediction probability. This pipeline is tuned jointly to maximize the classification accuracy on the training sets [14, 17, 18, 26, 30, 44]. However, one-hot vector makes no assumption about the similarity of words, and it is also very high-dimensional [3, 24].

RNNs improve time complexity and analyze texts word-by-word, then preserve the semantic of all of the previous text in a fixed-sized hidden layer [61].

The capability to capture superior, appropriate statistics could be valuable to capture the semantics of a long text in an RNN. However, an RNN is a biased model; recent words are more significant than earlier words. Therefore, they key components could appear anywhere across the document, not only at the end.

This might reduce the efficiency when used to capture the semantics of a whole document. Therefore, the long short-term memory (LSTM) model was introduced to overcome the difficulties of the RNN [60, 85].

A standard RNN makes predictions based only on considering the past words for a specific task. This technique is suitable for predicting the next word in context. However, for some tasks, it would be efficient if we could use both past and future words in tagging a task, as part-of-speech tagging, where we need to assign a tag to each word in a sentence [28].

In this case we already know the sequence of the words, and for each word we want to take both words to the left (past) and to the right (future) into consideration when making a prediction. That is exactly what the Bidirectional Neural Network (BNN) does; it consists of two LSTMs. One runs forward from left to right, and the other runs backward from right to left. This technique is successful in tagging tasks and for embedding a sequence into a fixed-length vector [30].

2.5 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) were initially designed for computer vision [41, 48]. CNNs exploit layers with convolving filters that are applied to local features, as shown in Figure 2.3.

CNNs reached outstanding results in computer vision where handcrafted features were used, e.g. scale-invariant features transform (SIFT) followed by a classifier; the main idea is to consider feature extractors and classifiers as one jointly trained task [6, 17]. The use of neural networks inspired many researchers after the successful approaches in [4, 5, 13].

This area has been investigated in recent years, especially by using multi-convolutional and pooling layers in CNNs and then sequentially extracting hierarchical representations of the input. CNN models for NLP achieved excellent results in semantic parsing [38], sentence modeling [43], search query retrieval [37], and other NLP tasks [5].

Recently, the DNN-based model has shown very good results for several tasks in NLP [6, 10, 17, 29, 30]. Despite the good performance of these models, in practice

they are relatively slow at training and testing, which restrains them from using a large scale of data, and it requires stacking many convolutional layers in order to capture long-term dependencies.

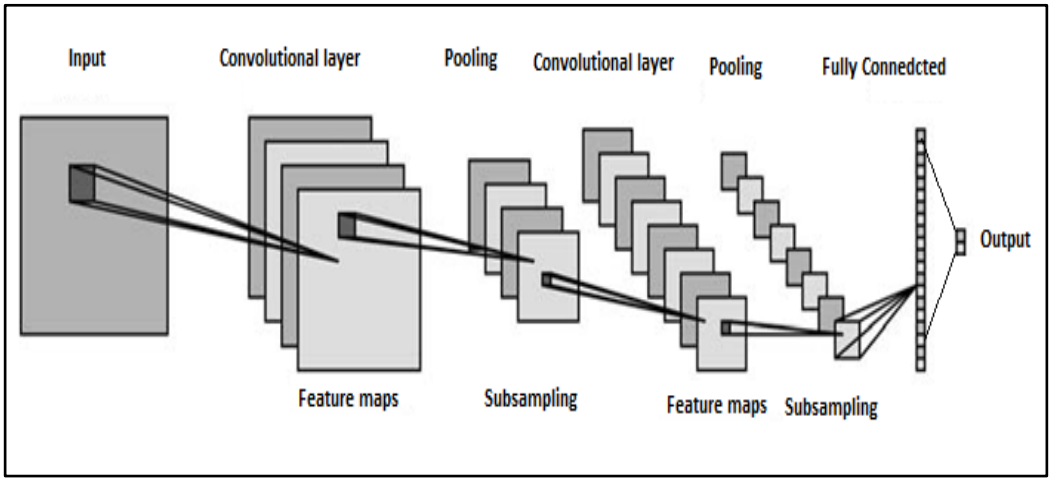

Figure 2.3. The standard structure of a CNN.

2.5.1 Pooling Layer

The pooling layers are a key aspect of the CNNs, typically applied after the convolutional layers. Pooling layers subsampling their input. Max-pooling is one of the most common way to apply pooling to the feature maps which are the result of each filter [48].

By performing max pooling in CNN NLP tasks, we are keeping information about whether the most important feature appeared in the sentence or not. However, we are losing global information about locality, and where in sentence something happened [29, 30].

Pooling provides fixed size output, which is essential for classification. Pooling usually decreases the output dimensionality and preserve most relevant feature.

The drawbacks of applying the pooling layers in CNN for NLP tasks, by performing the max pooling we are only keeping information about the most important feature appeared in the sentence, and we are losing information about where exactly it appeared.

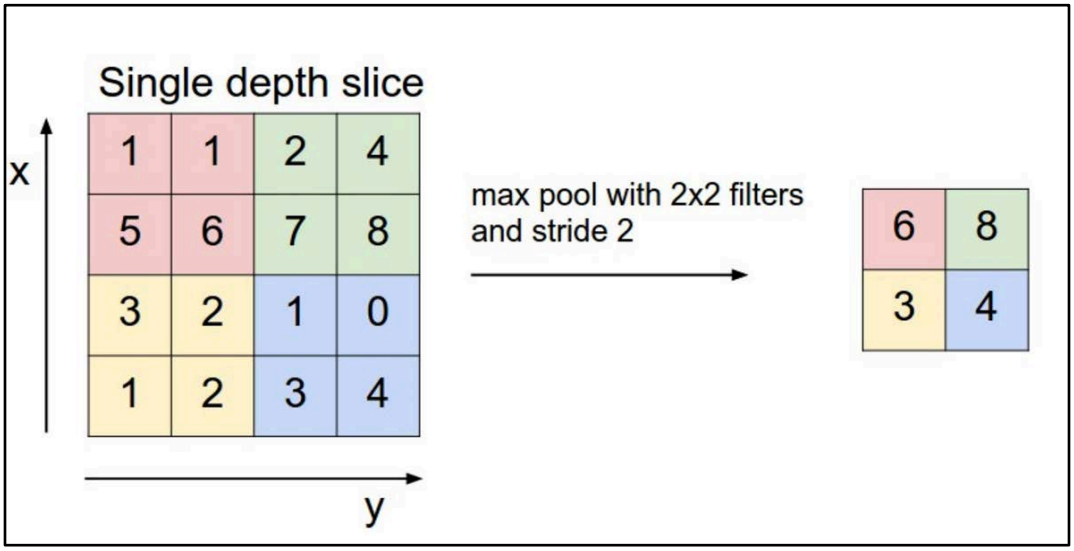

Figure 2.4. Max pooling in CNN [48].

The information locality is very important and that is similar to what a bag of n-grams model is doing. We lose global information about locality, and where in sentence something happens, but we are keeping local information captured by our filters.

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation as in Figure 3.4.

General pooling. In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

2.6 Convolution Neural Networks for Natural Language Processing (CNNs-NLP)

Convolutional Neural Networks (CNNs) initially designed for computer vision [41, 48]. CNNs exploit layers with convolving filters that applied to local features, CNNs reached an outstanding results in computer vision where handcrafted features were used, e.g. scale-invariant features transform (SIFT) followed by classifier; the main idea is to consider features extractors and classifier as one jointly trained task.

The use of neural network inspired many researchers after success approach in [5], and this area has been investigated in the recent years, especially by using multi convolutional and pooling layers, then sequentially extract hierarchical representation of the input.

CNNs models for NLP achieved excellent results in semantic parsing [38], sentence modeling [43], search query retrieval [44], and other NLP tasks [5]. Recently CNNs were applied to NLP systems and accomplished very interesting results [17, 29, 44]; convolutional layers are similar to a sliding window over a matrix. CNNs are numerous layers of convolutions with nonlinear activation functions, such as ReLU or tanh, applied to the results [86]. As shown in Figure 2.5.

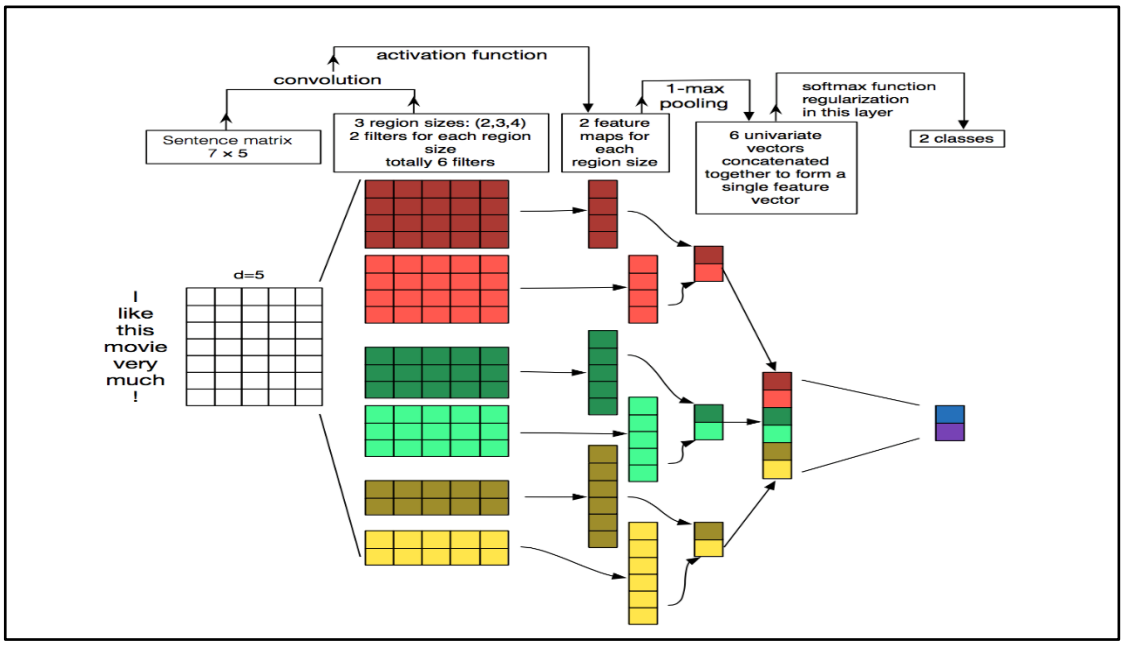

Figure 2.5. Convolutional Neural Network Architecture for NLP [86].

In a classical, feed-forward neural network, each input of a neuron is attached to each output in the next layer. This is called a fully connected or affine layer. However, CNNs have different approaches where they use convolutions over the input layer to compute the output. Local connections compute the output over the input layer, and then each layer applies different kernels, usually hundreds or thousands of filters, to then combine their results. During pooling or subsampling layers and during the training stage.

CNNs learn the values of their filter size based on the tasks. For instance, in image classification [41] a CNN might learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to detect higher-level features, such as facial shapes, in higher layers. The layer is then fed to a classifier that uses these high-level features. However, how does this apply to NLP?

As an alternative to image pixels, the input to most NLP tasks consists of sentences and documents represented as a matrix. Additionally, each row of the matrix matches up to one token, usually a word or character. Each row is a vector that represents a word.

Typically, this vector is a word-embedded, low-dimensional representation (e.g. word2vec, one-hot vectors) that indexes the word into a vocabulary (e.g. a ten word sentence using a 100-dimensional embedding, 10*100 matrix) as our input. In NLP, a filter slides over full words of the matrix. Therefore, the width of the filters is same as the width of the input matrix. Moreover, the region size may vary, but it is usually a sliding window over two to five words at a time.

2.7 GoogLeNet: Inception Convolution Neural Networks

The objective of the Inception architecture is created on finding out how an optimal local sparse structure in a convolutional vision network can be estimated and covered by readily available dense components. Note that assuming translation invariance means that our network will be built from convolutional building blocks. All we need is to find the optimal local construction and to repeat it spatially.

[87] Proposed a layer-by layer construction in which one should analyze the correlation statistics of the last layer and cluster them into groups of units with high correlation.

These clusters form the units of the next layer and are connected to the units in the previous layer. We assume that each unit from the earlier layer corresponds to some region of the input image and these units are grouped into filter banks.

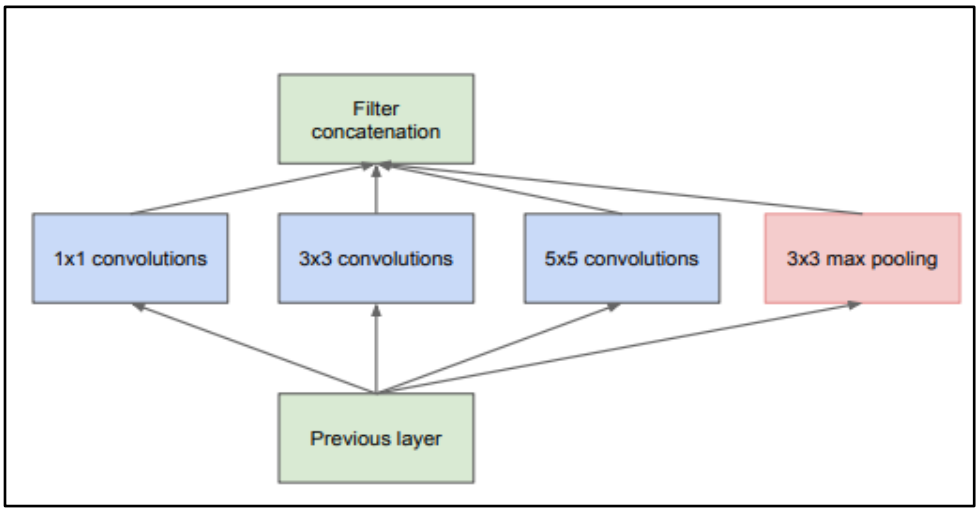

Figure 2.6.(a) Inception module, naïve version [88].

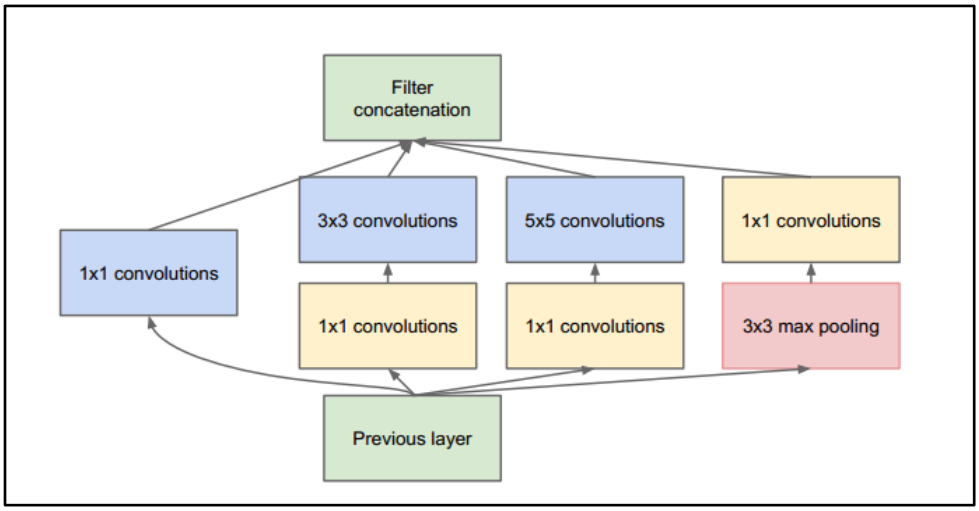

Figure 2.6.(b) Inception module with dimension reductions [88].

In the lower layers (the ones close to the input) correlated units would concentrate in local regions. This means, we would end up with a lot of clusters concentrated in a single region and they can be covered by a layer of 1×1 convolutions in the next layer, as suggested in [89]. However, one can also expect that there will be a

smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches, and there will be a decreasing number of patches over larger and larger regions.

In order to avoid patch alignment issues, current incarnations of the Inception architecture are restricted to filter sizes 1×1, 3×3 and 5×5, however this decision was based more on convenience rather than necessity.

It also means that the suggested architecture is a combination of all those layers with their output filter banks concatenated into a single output vector forming the input of the next stage. Additionally, since pooling operations have been essential for the success in current state of the art convolutional networks, it suggests that adding an alternative parallel pooling path in each such stage should have additional beneficial effect, too, see Figure 2.6.(a).

As these “Inception modules” are stacked on top of each other, their output correlation statistics are bound to vary: as features of higher abstraction are captured by higher layers, their spatial concentration is expected to decrease suggesting that the ratio of 3×3 and 5×5 convolutions should increase as we move to higher layers.

One big problem with the above modules, at least in this naıve form, is that even a modest number of 5×5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters. This problem becomes even more pronounced once pooling units are added to the mix: their number of output filters equals to the number of filters in the previous stage.

The merging of the output of the pooling layer with the outputs of convolutional layers would lead to an inevitable increase in the number of outputs from stage to stage.

Even while this architecture might cover the optimal sparse structure, it would do it very inefficiently, leading to a computational blow up within a few stages.

This leads to the second idea of the proposed architecture: carefully applying dimension reductions and projections wherever the computational requirements would increase too much otherwise.

This is based on the success of embeddings: even low dimensional embeddings might contain a lot of information about a relatively large image patch. However, embeddings represent information in a dense, compressed form and compressed information is harder to model. We would like to keep our representation sparse at most places as required by the conditions of [87], and compress the signals only whenever they have to be aggregated.

That is, 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose. The final result is depicted in Figure 2.6. (b).

In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion.

This is not strictly necessary, simply reflecting some infrastructural inefficiencies in our current implementation.

One of the main beneficial aspects of this architecture is that it allows for

increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity.

The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.

The improved use of computational resources allows for increasing both the width of each stage as well as the number of stages without getting into computational difficulties. Another way to utilize the inception architecture is to create slightly inferior, but computationally cheaper versions of it.

We have found that all the included the knobs and levers allow for a controlled balancing of computational resources that can result in networks that are $2 - 3\times$ faster than similarly performing networks with non-Inception architecture, however this requires careful manual design at this point.

2.8 Recurrent Neural Networks (RNNs)

Recurrent neural network (RNN) improved time complexity; in this model text is analyzes word by word then preserve the semantic of all the previous text in a fixed-sized hidden layer [61]. As shown in Figure 2.7. The capability to capture superior appropriate statistics could be valuable to capture semantics of long text in recurrent network. However, recurrent network is biased model, because recent words more

significant than earlier words. Therefore, the key components could appear anywhere across the document not only at the end.

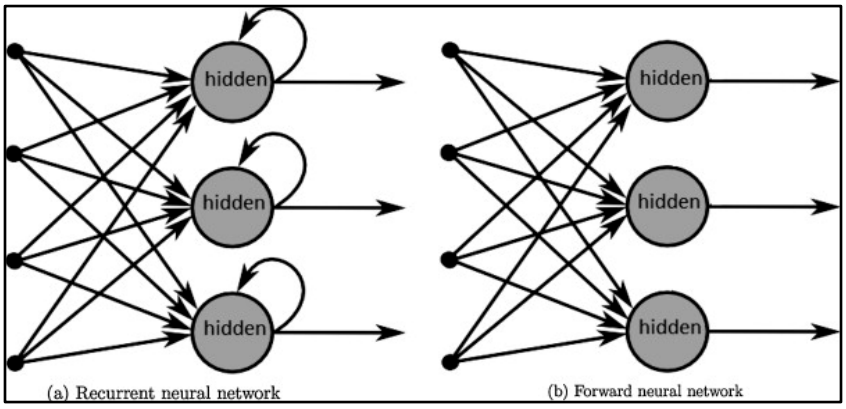

Figure 2.7. Recurrent Neural Network Vs Traditional Neural Network [90].

This might reduce the efficiency when used to capture the semantic of whole document. Long short-term Memory (LSTM) model introduced to overcome the difficulties of the RNN [60, 85].

A standard RNN makes prediction based only on considering the past word into account for specific task this technique is create for predicting the next word in context. However, for some task it would be efficient if we could use both past and future for instance, tagging task, like part-of-speech tagging, where we need to assign a tag to each word in a sentence [91].

In this case we know already all the sequence of the words, and for each word we want to take both words to the left (past) and words to the right (future) in consideration when we want our prediction.

That is exactly what Bidirectional neural network does; it consist of two LSTM one runs forward from left to right, and the other one run backward from right to left.

This technique is success in tagging tasks and for embedding a sequence into a fixed-length vector [29, 90].

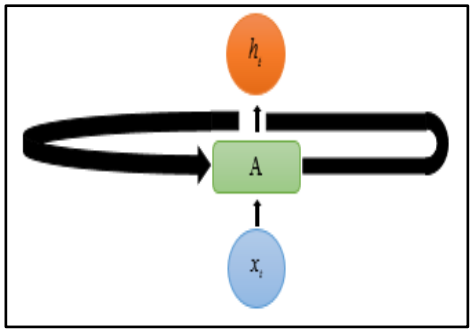

Figure 2.8. Recurrent Neural Network with loops.

The intuition of RNNs is that human's do not start their thinking from scratch every second, the objective of the recurrent RNNs is to make use of sequential information, the output is based on the previous computation.

In traditional RNNs all input are independent of each other; while this approach is inefficient for many task in NLP (e.g. predicting the next word in a sentence) in this case it is important to know the previous word in order to predict the next word in the context. RNNs have shown great success in many NLP tasks [22, 44, 85, 91, 92].

RNNs have a memory which captures information in arbitrary long sequences. In Figure 2.8, a chunk of neural network, looks at input and outputs a value. A loop allows information to be passes from one step of the network to the next step.

RNNs is a type of deep neural networks that are deep in temporal dimension and it has been used widely in time sequence modeling.

The objective behind RNNs for sentence embedding is to find a dense and low dimensional semantic representation by recurrently and sequentially processing each

word in a sentence, and mapping it into a low dimensional vector.

The global contextual feature of the whole text will be in the semantic representation of the last word in the sequence [21, 22, 93]. We also can think of RNNs as multiple copies of the same network, where each one is passing a message to inheritor. What will happens if we unroll the loop, as shown in Figure 2.9.

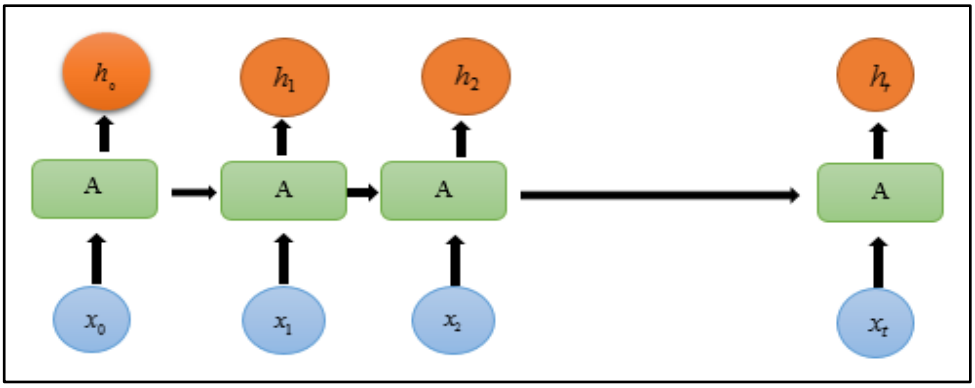

Figure 2.9. An unrolled RNNs.

We can compute the output o_t as follows in simple RNN:

$$o_t = f(W_o h_t) \tag{2.6}$$

$$h_t = \sigma(W_h h_{t-1} + W_x x_t) \tag{2.7}$$

Where $W_o, W_h,$ and W_x are the matrices for hidden layer output h_t, past hidden layer activity h_{t-1} and the input x_t.

The time recurrence is presented in Eq. (2) which conveys the present hidden layer activity $h_t$with its past hidden layer activity h_{t-1}. this reliance is nonlinear due to using of logistic function $\sigma(.)$.

2.8.1 Recurrent Neural Networks Based Language Models (RNNLMs)

Recurrent neutral network language model (RNNLM) proposed in [65] and extension in [66]. The main difference between feedforward and the recurrent architecture is in representation of history, while for feedforward NNLM, the history is still just previous several words, for the recurrent model an effective representation of history is learned from the data during training.

The hidden layer of RNN represents all previous history and not just $n-1$ previous words, thus the model can theoretically represent long context patterns.

Another important advantage of the recurrent architecture over the feedforward one is the possibility to represent more advanced patterns in the sequential data. For example, patterns that rely on words that could have occurred at variable position in the history can be encoded much more efficiently with the recurrent architecture - the model can simply remember some specific word in the state of the hidden layer, while the feedforward architecture would need to use parameters for each specific position of the word in the history; this not only increases the total amount of parameters in the model, but also the number of training examples that have to be seen to learn the given pattern [60, 85, 94-96].

The architecture of RNNLM is shown in Figure 2.10. The input layer consist of a vector $w(t)$ that represents the current word w_t encoded as 1 of V (thus size of $w(t)$ is equal to the size of the vocabulary), and of vector $s(t-1)$ that represents output values in the hidden layer from the previous time step. After the network is trained, the

output layer $y(t)$ represents$P(w_{t+1}|w_t, s(t-1))$. The network is trained by stochastic gradient decent using either usual backpropagation (BP) algorithm, or backpropagation through time (BPTT) [65].

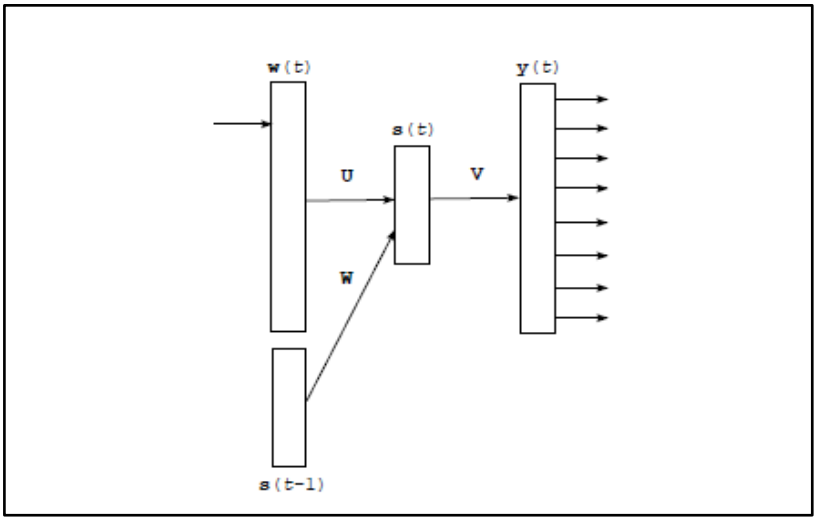

Figure 2.10. Simple recurrent neural network [66].

The network is represented by input, hidden and output layers and corresponding weight matrices; matrices U and W between the input and the hidden layer, and matrix V between the hidden and the output layer.

2.8.2 The Problem of Long-Term Dependencies

One of the appeals of RNNs is the idea that they might be able to connect precious information to the present task, such as using previous video frames might inform the understanding of the present frame.

If the RNNs could do this, they would be extremely useful. But can they? It depends. Some time we only need to look at recent information to perform the present task. For instance, consider a language model trying to predict the next word based on

the previous one. If we are trying to predict the last word we do not need any further context. Sometime the gap between the relevant information and the place that it is needed is small, RNNs can learn to use the past information.

In classic RNNs during the gradient back propagation stage, the gradient signal can end up being multiplied multiple of times (as many as the number of the time steps) by the weight matrix; that linked with the connection between the neurons of the recurrent hidden layer. As shown in Figure 2.11.

Which means that, the volume of weights in the transition matrix can have a significant impact on the learning process. If the weights matrix are small meaning that the leading eigenvalue of the weight matrix is smaller than one, then it can cause the vanishing gradients problem, where the gradient signal become very small then learning become very slow or stops working.

Furthermore, it can make the task of learning long-term dependencies more difficult in the data. On the other hand, if the weights in this matrix are larger meaning the leading eigenvalue of the weight matrix is larger than one, then it can lead to situation where the gradient signal is very large, and then that can cause learning to swerve; this called exploding gradients problem [92].

RNNs consider as deep neural networks across many time instance, the gradient of a sentence may not be able to back-propagate to the beginning of a sentence, and that is due to many of nonlinearity transformations [29, 61, 70].

RNNs might be able to link previous information to the present task, such as using the earlier video frames might inform the understanding of the current frame. Sometimes we need to look at a latest information to perform the current task. It is

possible for the gap between the relevant information and the point where is needed to become very large. And if that gap grows, then RNNs become unable to learn to connect the information.

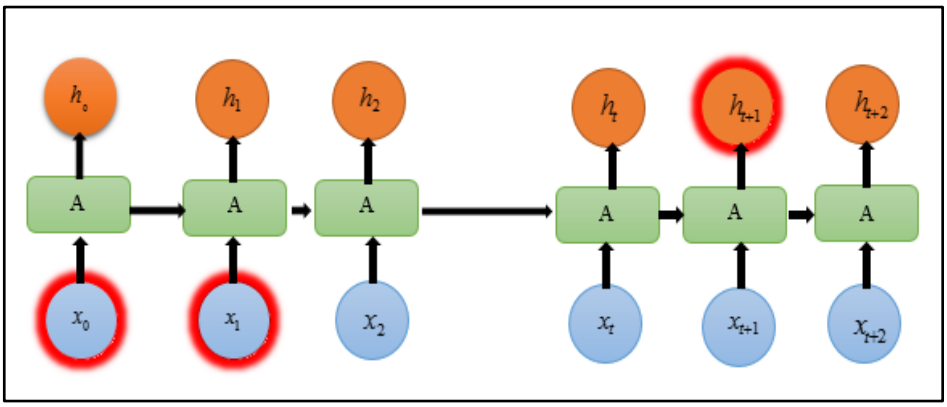

Figure 2.11. Long-term dependences problem in RNNs.

LSTM is an extension of RNN [33, 97] instead of using the nonlinear connection between the past hidden activity and the current layer hidden activity, it uses a linear dependence to relate its past memory to the current memory. Most important, in LSTM the forget gate was presented to restrain each element of the past memory to be contributed to the current memory cell.

2.8.3 Vanishing and Exploding Gradients

By the early 1990s, the vanishing gradient problem emerged as a major obstacle to recurrent network performance [92].

Just as a straight line expresses a change in x alongside a change in y, the gradient expresses the change in all weights with regard to the change in error. If we cannot know the gradient, then we cannot adjust the weights in a direction that will decrease error, and our network stops to learn.

Recurrent nets looking for to establish connections between a final output and events many time steps before were hobbled, because it is very difficult to know how much importance to accord to remote inputs [48, 60].

This is partially because the information flowing through neural nets passes through many stages of multiplication, and because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing or exploding.

Exploding gradients treat every weight as though it were the proverbial butterfly whose flapping wings cause a distant hurricane. Those weights' gradients become saturated on the high end; i.e. they are presumed to be too powerful. But exploding gradients can be solved relatively easily, because they can be truncated or squashed. Vanishing gradients can become too small for computers to work with or for networks to learn – a harder problem to solve. The data is flattened until, for large stretches, it has no detectable slope. This is analogous to a gradient vanishing as it passes through many layers.

2.9 Long Short-Term Memory (LSTM)

Long Short Term Memory network, usually called (LSTM), are special kind of RNN, capable of capturing learning long-term dependencies [85]. The memory cell is consist of four main components: Input gate, Memory cell, Forget gate, Output gate. The self-recurrent connection has a weight of one, the state of the memory cell can remain constant from one time-step to another.

The gates assist to control the interfaces between the memory cell itself and its

environment, input gate can allow incoming signal to alter the state of the memory cell or block it, the output gate can allow the state of the memory cell to have an effect on other neurons or prevent it. Finally, the forget gate can control the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state as needed [60].

The traditional RNN we described above is hard to train due to the gradient vanishing and exploding problems, which is due to the nonlinear relation between h_t and h_{t-1}, LSTM introduces a linear dependence between the memory cell c_t and its past c_{t-1} . Furthermore, LSTM has input and output gates applied on non-linear function. LSTM is describe as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1}) \tag{2.8}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1}) \tag{2.9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1}) \tag{2.10}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t) \tag{2.12}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.13}$$

Where i_t, f_t and o_t are the input gate, forget and output gate. The core idea of LSTM is the cell state, it run straight down the entire chain.

LSTM able to add and remove information to the cell state regulated by gates, which are composed of sigmoid layer, the sigmoid layer produce numbers between zero and one, zero means "stop anything to go through", while one means "let everything through".

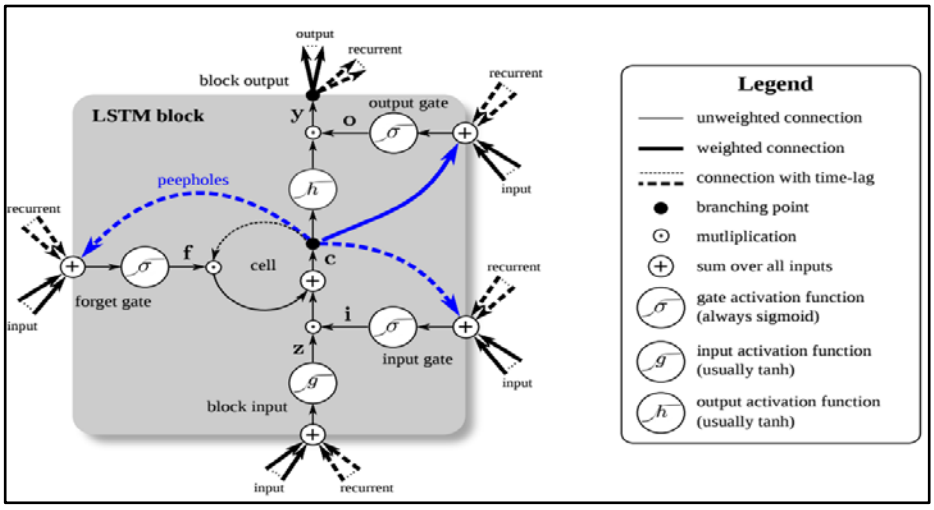

Figure 2.12. Detailed Schematic of Recurrent Network Long Short-Term Memory block [96].

LSTM allows preservation of gradients. The memory cell remembers the first inputs as long as the forget gate is open and the input gate is closed. The output gate provides finer control to switch the output layer on or off without altering the cell contents. As described in Figure 2.12.

2.10 Bidirectional Recurrent Neural Networks (BRNNs)

Bidirectional Recurrent Neural Networks (BRNNs) [98, 99] do not require their input data to be fixed and their future input information is reachable from the current state. One property of the recurrent layer is that there is imbalance in the amount of information seen by the hidden states at different time steps.

The objective is to connect two hidden layers of opposite directions to the same output. The output layer can get information from past and future states. The earlier hidden states only observe a few vectors from the lower layer, while the later ones are computed based on most of the lower-layer vectors.

BRNN is composed of two recurrent layers working in opposite directions, which will return two sequences of hidden states from the forward and reverse recurrent layers, respectively.

$$H_{forward} = (h_1^{\rightarrow}, h_2^{\rightarrow}, \dots, h_{T'}^{\rightarrow}) \tag{2.14}$$

$$H_{reverse} = (h_1^{\leftarrow}, h_2^{\leftarrow}, \dots, h_{T'}^{\leftarrow}) \tag{2.15}$$

Then we take the last hidden states of both directions and concatenate them to form a fixed-dimensional vector:

$$h = [h_{T'}^{\rightarrow}; h_1^{\leftarrow}] \tag{2.16}$$

Finally, the fixed-dimensional vector h is fed into the classification layer to compute the predictive probabilities $p(y = k|X)$ of all the categories $k = 1, \dots, K$ given the input sequence X.

2.11 Gated Recurrent Unite (GRU)

Gated Recurrent Unit (GRU) is a RNNs variant proposed in [18], the network combines the forget and input gates into a single update gate, also merge cell state and hidden state, it is similar to LSTM using gating functions, the GRU does not have a memory cell, GRU operation can be describe as following:

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \tag{2.17}$$

$$z_t = \sigma(W_z x_t - U_z h_{t-1}) \tag{2.18}$$

$$\widetilde{h_t} = \tanh(W_h x_t + U(r_t \odot h_{t-1})) \tag{2.19}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2.20}$$

Where the output from GRU is h_t, z_t and r_t are the update gate and rest gate.

$\tilde{h}$ is the candidate output. W_z, W_h, W_r, U_z, And U_r are the matrices in GRU.

2.12 Vector Representations of Words

Word2vec model proposed by [14, 26]. This model is used for learning vector representations of words, called (word embedding). Natural language processing systems traditionally treat words as discrete atomic symbols, and therefore 'cat' may be represented as Id537 and 'dog' as Id143. These encodings are arbitrary, and provide no useful information to the system regarding the relationships that may exist between the individual symbols.

This means that the model can leverage very little of what it has learned about 'cats' when it is processing data about 'dogs' (such that they are both animals, four-legged, pets, etc.). Representing words as unique, discrete ids furthermore leads to data sparsity, and usually means that we may need more data in order to successfully train statistical models. Using vector representations can overcome some of these obstacles.

Vector space models (VSMs) represent words in a continuous vector space where semantically similar words are mapped to nearby points, and embedded nearby each other. VSMs have a long, rich history in NLP, but all methods depend in some way or another on the Distributional Hypothesis, which states that words that appear in the same contexts share semantic meaning.

The different approaches that leverage this principle can be divided into two categories: count-based methods (e.g. Latent Semantic Analysis) [100], and predictive methods (e.g. neural probabilistic language models) [20].

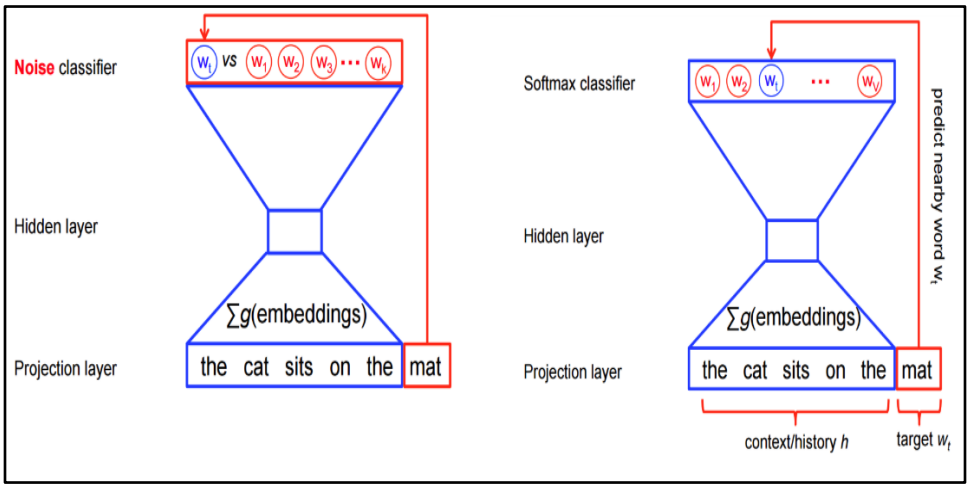

Figure 2.13. The CBOW and Skip-gram models [14].

This distinction is elaborated in much more detail by [101], but in a nutshell: Count-based methods compute the statistics of how often some word co-occurs with its neighbor words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word.

Predictive models directly try to predict a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model).

Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text. It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model [14]. Algorithmically, these models are similar, except that CBOW predicts target words (e.g. 'mat') from source context words ('the cat sits on the'), while the skip-gram does the inverse and predicts source context-words from the target words.as illustrated in Figure 2.13.

This inversion might seem like an arbitrary choice, but statistically it has the effect that CBOW smoothest over a lot of the distributional information (by treating an

entire context as one observation).

For the most part, this turns out to be a useful thing for smaller datasets. However, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

2.13 Combination of Convolution Neural Networks and Recurrent Neural Networks (CNNs-RNNs)

The combination of both CNNs and RNNs is explored for speech recognition [27], and a similar approach was applied to image classification [41]. [29] Investigated the combination of CNN-RNN to encode character input, and implemented a high-level feature input sequence of character level to capture sub-word information. However, this model performs best when a large number of classes are available.

[69] Outlined structured attention networks, which incorporate graphical models to generalize simple attention, describe the technical machinery and computational techniques for backpropagation through models of this form.

[77] aimed to improve representation efficiency, and the model employed Differential State Framework (DSF). DSF models maintain longer-term memory by learning to interpolate between a fast-changing, data-driven representation and a slowly changing, implicitly stable state.

[11] Investigated an approach to advance the accuracy of the deep learning method for sentiment analysis by incorporating domain knowledge; this paper combined domain knowledge with deep learning, using sentiment scores learnt by regression to augment the training data. They also utilized weighting across entropy

with a penalty matrix as an enhanced loss function.

We observed that the use of a vanilla CNN for text classification has one drawback. In [29] the network must have many layers in order to capture long-term dependencies in an input sentence. Perhaps that might be the motivation behind [6], which utilized a very deep convolutional network with several convolutional layers followed by two fully connected layers.

CHAPTER 3: RESEARCH PLAN

3.1 Deep Neural Network Language Model for Text Classification

In this section, we present the details of the proposed model is shown in Figure 3.1, which consists of convolutional and recurrent neural networks. Our model's architecture uses word embeddings as inputs and takes them to a convolutional neural network to learn to extract high-level features, whose outputs are then given to a long short-term memory recurrent neural network language model to assets the model to capture long term dependencies, then finally followed by a classifier layer.

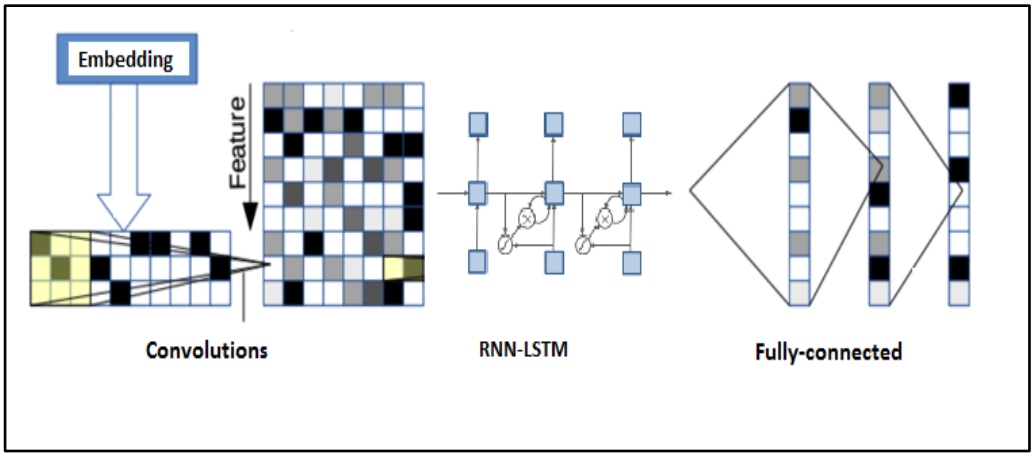

Figure 3.1. The proposed CNN-LSTM architecture

3.2 The Embedding Layer

The first layer of the network transforms words into real-valued feature vectors

that capture semantic and syntactic information. Our model's input is a sequence of words $[w_i, \dots, w_s]$, with each word being derived from vocabulary V. Words are denoted by distributed vector $W \in R^{1*d}$ and looked up in a word embedding matrix $W \in R^{1*|V|}$. This is formed by simply concatenating embeddings of all words in V.

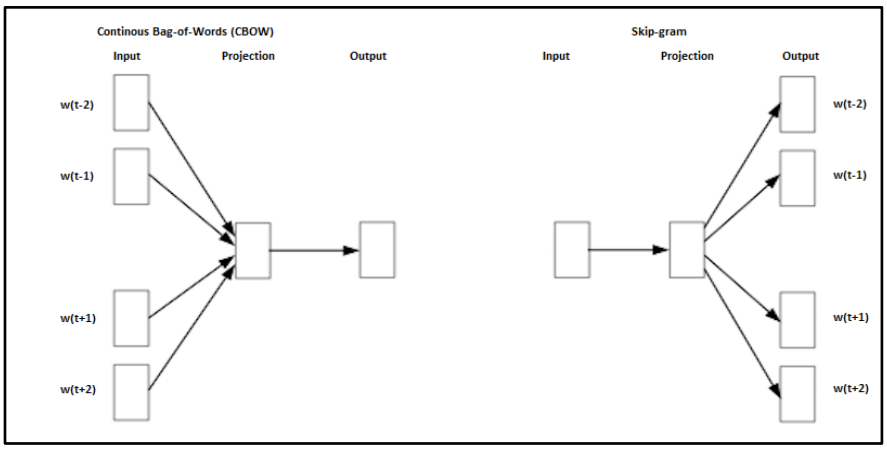

Figure 3.2. Architecture for the CBOW and Skip-gram [14].

We initialized the model using word vectors obtained from unsupervised neural language model which is a popular method to improve performance in the absence of a large supervised training set. We exploits word2vec that were trained on 100 billion words from Google News [14, 26].

Words not existent are initialized randomly; initializing the word vectors with pre-trained vectors obtained from an unsupervised neural language model is a very successful method [17].

It can capture syntactic and semantic information, which are very important for sentiment analysis task. In our work, we preform unsupervised learning of word-level

embeddings using word2vec tool [14], that implement skip-gram and continuous bag-of-words architectures for computing vector representations of words as shown in Figure 3.2.

3.3 The Convolutional layer

The model shown in Figure 3.3 is a slight variant of the CNN architecture of [5]. Let $x_i \in \mathbb{R}^k$ to be the k-dimensional word vector corresponding to the ith word in the sentence of length n, which is represented as:

$$x_{1:n} = x_1 \oplus x_2 \dots \oplus x_n, \tag{3.1}$$

Where $\oplus$ is the concatenation operator, overall let $x_{i::i+j}$ refer to the concatenation of words $x_i, x_{i+1}, \dots x_{i+j}$. The convolutional operational consists of a filter$w \in \mathbb{R}^{hk}$, which is applied to a windows of h words to produce a new features. For instance, a feature is c_i is generated from a window of words $x_{i:i+h-1}$ by:

$$c_i = f(w.x_{i:i+h-1} + b). \tag{3.2}$$

Where $b \in \mathbb{R}$ a bias is term and f is a non-linear function such as the hyperbolic tangent. This filter is applied to each possible window of words in the sentence $\{x_{1:h}, x_{2:h+1}, \dots, x_{n-h+1:n}\}$ to produce a feature map of:

$$c = [c_1, c_2, \dots, c_{n-h+1}], \tag{3.3}$$

With $c \in R^{n-h+1}$. Then, we fed the feature maps to a recurrent layer LSTM in order to capture long-term dependencies. This technique will reduce the number of parameters in the proposed model.

Max-over-time pooling operation was not applied, we argue that the pooling

layer is the reason for lost details in local information, because the pooling layer only captures the most important feature in a sentence and ignored the others; therefore, we attempt to exclude the pooling layer and utilize it with a recurrent layer to assist the model to capture long-term dependencies more efficiently and reduce the number of the parameters in the proposed architecture; we fed the feature map into single layer of LSTM.

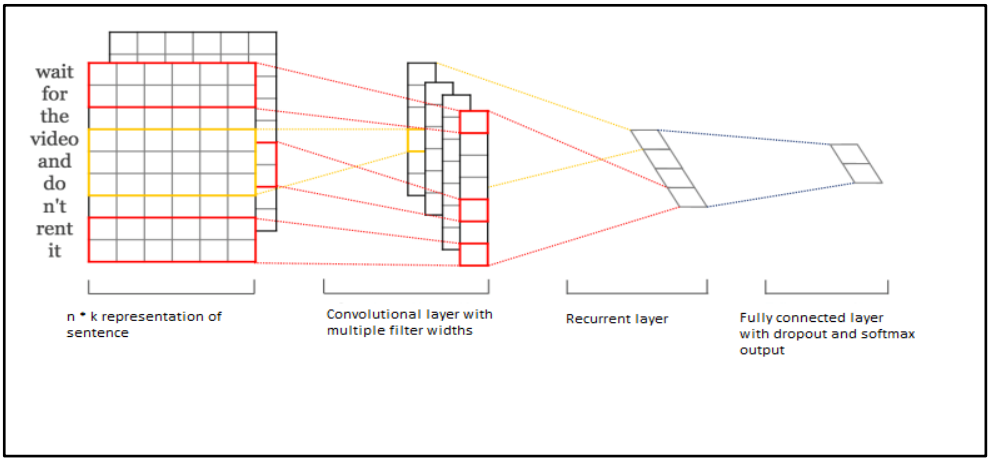

Figure 3.3. Conv-Lstm Model for NLP

3.4 The Recurrent Layer

The objective of the RNN is to make use of sequential information, and the output is based on the previous computation. All inputs are independent of each other in traditional neural network, while this approach is inefficient for many tasks in NLP (e.g. predicting the next word in a sentence) in this case it is important to know the previous word. RNN has a memory that capture information in arbitrary long sequences, which is illustrated in Fig 3.4.

$$h_t = f(x_t, h_{t-1}), \tag{3.4}$$

Where $x_t \in \mathbb{R}^d$ one time step from the input sequence, $(x_1, x_2, \dots, x_T)$. $h_o \in \mathbb{R}^d$ often initialized as an all-zero vector.

Recursive neural networks proved to be efficient for constructing sentence representations. The model has a tree structure which is able to capture the semantic of sentence. However, this is a time-consuming task due to constructing the textual tree complexity [28].

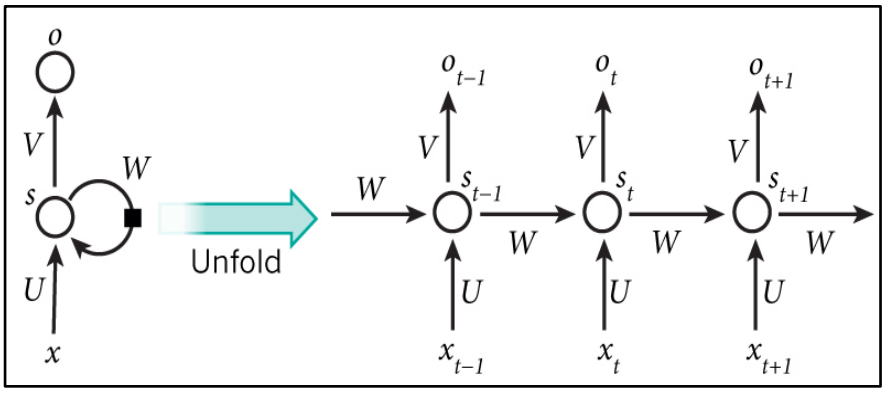

Figure 3.4. RNN unfold framework [73].

Recurrent neural network has enhanced time complexity. In this model, text is analyzed word by word and then preserves the semantic of all the previous text in a fixed-sized hidden layer [61].

The capability to capture superior appropriate statistics could be valuable for capture semantics of long text in recurrent networks. However, recurrent networks is biased model, because recent words are more significant than earlier words. Therefore, the key components could appear anywhere across the document and not only at the end; this might reduce the efficiency when used to capture the semantic of the whole

document. The LSTM model was introduced to overcome these difficulties.

The most naïve recursive function is known to be suffer from the problem of vanishing gradient. More recently it is common to use Long Short-Term Memory LSTM [85, 94]. RNN in Figure 3.4 is a type of neural network architecture specially used for sequence modeling. At each time step t, a recurrent layer takes the input vector $x_t \in R^n$ and hidden state h_t by applying the recursive operation:

$$h_t = f(W_{xt} + Uh_{t-1} + b) \tag{3.5}$$

Where $W \in R^{m*n}, b \in R^{m*m}, b \in R^m$ parameters, and f is an element-wise nonlinearity. Learning long-term dependencies with a vanilla RNN is difficult because of the vanishing and exploding gradient [92].

The Long short-term memory LSTM [60, 85] overcomes the deficiencies of the vanilla RNNs by augmenting the RNNs with a memory cell that takes as an inputx_t, h_{t-1}, c_{t-1}, and produces h_t, c_t by the following:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i) \tag{3.6}$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \tag{3.7}$$

$$O_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \tag{3.8}$$

$$\mathbb{g}_t = \sigma(W^{\mathbb{g}} x_t + U^{\mathbb{g}} h_{t-1} + b^{\mathbb{g}}) \tag{3.9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \mathbb{g}_t \tag{3.10}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.11}$$

Where σ, and $tanh$ are the element-wise sigmoid and hyperbolic tangent function and i_t, f_t, o_t are referred to as input, forget, and output gates. At $t = 1, h_0, c_0$ are initialized to zero vector. $\odot$ The element-wise multiplication operator. Parameters of the LSTM are preservative with respect to time.

LSTM outperforms vanilla RNNs on many tasks, including language modeling [44]. It is easy to extend LSTM to more than one layer, having multiple layers is critical for attaining competitive performance on various tasks [21].

3.5 LSTM Layer

LSTM is more complicated function that learns to control the flow of information, to prevent the vanishing gradient and to allow the recurrent layer to more easily capture long-term dependencies. LSTM was initially proposed in [60, 85] and later modified in [21].

RNN has problems of gradient vanishing or explosion. Meanwhile, RNNs are considered as deep neural networks across many time instances. The gradient at the end of the sentence may not be able to back-propagate to the beginning of the sentence, because of the nonlinearity transformation [41, 61]. These problems are the main motivation behind the LSTM model, which introduces a new structure called a memory cell in Figure 3.5. The memory cell is consist of four main components: input, output, forget gates and candidate memory cell. The following equations describe how the memory cells layer are updated at every timestep t. First, we compute the values for i_t, the input gate, and $\tilde{c}_t$ the candidate value for the states of the memory cells at time t:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i \tag{3.12}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.13}$$

Given the new value of the input gate activation i_t, the forget gate activation f_t and the candidate state value $\tilde{c}_t$, we can compute c_t the memory cells new state at time t:

$$c_t = i_t * \tilde{c} + f_t * c_{t-1} \tag{3.14}$$

With the new state of the memory cells, we compute the value of their output gates and, subsequently, their outputs

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t + b_o) \tag{3.15}$$

$$h_t = o_t * \tanh(c_t) \tag{3.16}$$

Where x_t is the input to the memory cell layer at time t. $W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o$, and V_o are weight matrices. b_i, b_f, b_c, b_o, are bias vectors.

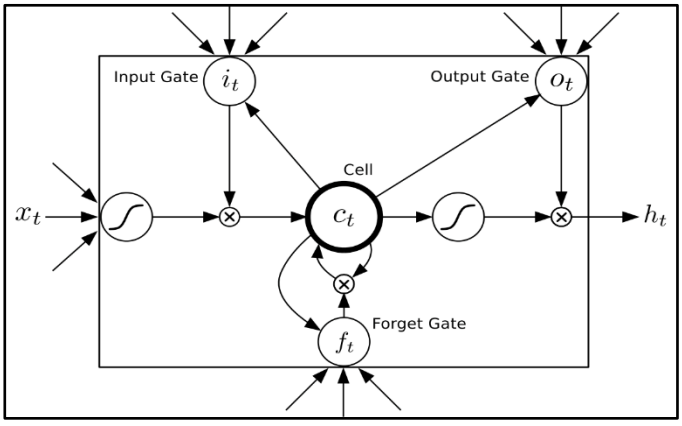

Figure 3.5. LSTM. Shown the five key architecture elements of LSTM [21].

3.6 Back Propagation through Time (BPTT)

Back propagation through time (BPTT) is the key algorithm that makes training

deep models computationally controllable, and it is a way of computing gradients of expression through the recursive application of the chain rule. The core issue we are given is some function $f(x)$ were x is the vector of inputs, and we are interested in computing the gradient of f at $x(i.e\nabla f(x))$.

Error can be even backpropagated further [31]. BPTT is a simple extension of the backpropagation algorithm for recurrent neural network; with BPTT the error is broadcasted via recurrent connection back in time for specific time steps. Therefore, the network absorbs and remembers information for numerous time steps in the hidden layer when it is learned by BPTT. More details about the implementation described can be found in [102].

3.7 Classification Layer

The classification layer is in principle, a logistic regression classifier. It gives a fixed-dimensional input from the lower layer; the classification layer affine transforms it, followed by a softmax activation function to compute the predictive probabilities for all the categories [102]. This done by:

$$p(y = k|X) = \frac{\exp(w_k^T x + b_k)}{\sum_{k=1}^{k} \exp(w_k^T x + b_k)} \tag{3.17}$$

Where w_k and b_k are the weight and bias vectors. We assume there are k categories. This classification layer takes as input a fixed-dimensional vector, while the recurrent layer or convolutional layer returns a variable-length sequence of vectors, this can be addressed by wither simply ,max-pooling the vector as over time dimension for

convolutional and recurrent [17] .

3.8 Unsupervised Learning of Word-Level Embedding

Initializing word vectors with those obtained from an unsupervised neural language model is a popular method to improve performance in the absence of a large, supervised training set [3, 80].

It has been recently shown that improvements in model accuracy can be obtained by performing unsupervised, pre-trained word embeddings. In our experiments, we utilized the publicly available word2vec vectors that were trained on 100 billion words from Google news.

The vectors were trained using a continuous bag-of-words algorithm [26]. While the word embeddings are obtained, the model captures syntactic and semantic aspects of the words they represent; however, they have no notion about their sentiment behavior.

Word embeddings play an important role in our neural language model. They are able to capture syntactic and semantic information, which are very significant to sentiment analysis.

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 SENTIMENT ANALYSIS DATASETS

The performance of the proposed model was evaluated on two benchmark sentiment analysis datasets: the Stanford Large Movie Review dataset (IMDB) and the Stanford Sentiment Treebank dataset (SSTb) [40], derived from Rotten Tomatoes movie reviews [103]. As shown in Table 4.1.

4.1.1 STANFORD LARGE MOVIE REVIEW DATASET (IMDB)

The Stanford Large Movie Review (IMDB) dataset was first proposed by [104] as a benchmark for sentiment analysis. It consists of 50,000 binary labeled reviews; the reviews are divided into 50:50 training and testing sets.

The distribution of labels with each subset of data is balanced. We used 15% of the labeled training documents as a validation set. One key aspect of this dataset is that each review has several sentences. .

The average length of each document is 241 tokens, with standard deviation of 198.8 tokens; the maximum length of a document is 2,526 words.

Dataset	Set	Sentence	Binary
SSTb	Train	8544	2, 5
	Dev	1101	2,5
	Test	2210	2, 5
IMDB	Train	2210	2
	Dev	4k	2
	Test	25k	2

Table 4.1 Sentiment Analysis Datasets.

4.1.2 STANFORD SENTIMENT TREEBANK DATASET (SSTb)

The Stanford Sentiment Treebank (SSTb) dataset was first proposed by [103] and extended by [40] as a benchmark for sentiment analysis. It consists of 11,855 reviews taken from the movie review site Rotten Tomatoes, with one sentence for each review.

The SSTb was split into three sets: 8544 sentences for training, 2210 sentences

for testing, and 1101 sentences for validation (or development). The SSTb also includes fine-grained sentiment labels. In Table 4.1, we present additional details about the two benchmark datasets.

4.2 EXPERIMENTAL SETUP

4.2.1 HYPERPARAMETERS AND TRAINNIG

We used stochastic gradient descent (SGD) to train the network and the back-propagation algorithm to compute the gradient. We believe that by adding a recurrent layer to the model as an alternative to the pooling layer, we can effectively reduce the number of the convolutional layers needed to capture long-term dependencies. Therefore, we consider merging a convolutional and recurrent layer into one single model.

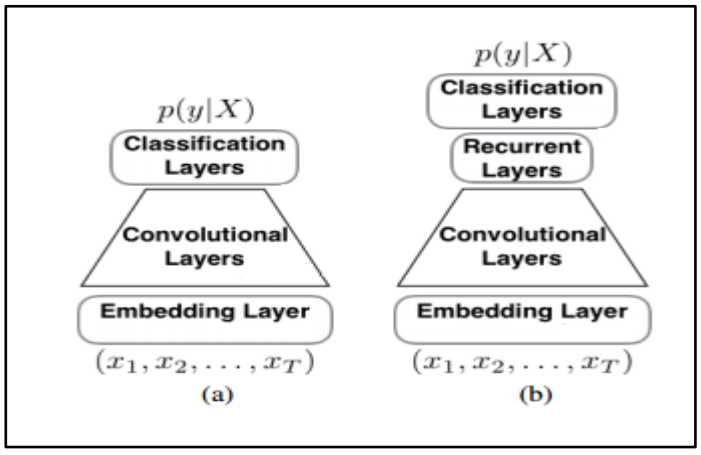

Figure 4.1. Graphical illustration of (a) the convolutional network and (b) the proposed convolutional-lstm Model for text classification

Our architecture goal is to reduce the need for stacking multiple convolutional and pooling layers in the network, as shown in Figure 4.1 in order to reduce the depth of the network and the loss of detailed, local information. Thus, in the proposed model. We consider convolutional layers with only one layer that has $d = 256$ filters and a receptive field size of r (3,3,5). For an activation function we use rectifier linear units in the convolutional layer (ReLU).

The recurrent layer is fixed to a single layer of LSTM. The hidden state dimension is $d = 128$. For both datasets, the number of training epochs varies between (5, 20).

We compared the proposed model with methods using word embedding and convolutional architecture and different deep learning and traditional methods.

We also focused on the regularization, the learning rate, and dropout parameters; we then extracted sentence features with the convolutional layer.

The recurrent layer provides an indication of the robustness of our approach in multiple domains. In Table 4.2, we show the selected hyperparameter value for the proposed architecture.

4.2.2 REGULARZATION

For regularization we employ dropout as an effective method to regularize deep neural networks and neural networks.

Dropout prevents co-adaption of hidden units. We apply it with constraint on the L2-norms of the weight vectors [105]; we insert dropout modules in between CNN and LSTM layers to regularize them i.e., setting zero a proportion p of the hidden units

during forward-backpropagation.

That is, given the penultimate layer $z = [\hat{c}_1, \ldots, \hat{c}_m]$ (here we have m filters).

Parameter	CNN	RNN-LSTTM
Word-Embedding-Dimension	300	300
Word Context Units	5	5
Hidden Units	-	-
Learning rate	0.01	0.01
Dropout	0.5	0.5

Table 4.2. Hyperparameter initialization ranges.

4.2.3 OPTIMIZATION

Training was done through stochastic gradient descent over shuffled mini-batches. For training and validation, we randomly split the full training examples. The size of the validation set is the same as the corresponding test size and is balanced in each class.

We trained the model by minimizing the negative log-likelihood or cross entropy loss. Early stopping was utilized to prevent overfitting. In our work, we employed unsupervised learning of word-level embedding using the word2vec, which implemented the continuous bag-of-words and skip-gram architectures for computing vector representations of a word.

We validated the proposed model on two datasets, considering the difference in the number of parameters. However, the accuracy of the model does not increase with the number of convolutional layers.

More pooling layers typically leads to the loss of long-term dependencies. Therefore, in our model we removed the pooling layer from the convolutional network and replaced it with a recurrent layer to reduce the loss of local information.

One recurrent layer is enough to capture long-term dependencies in the input sequence.

4.3 RESULTS AND ANALYSIS

4.3.1 ANALYSIS OF THE STANFORD SENTIMENT TREEBANK DATASET (SSTb)

For the Stanford Sentiment Analysis dataset (SSTb), we performed several experiments to offer a fair comparison with competitive models. We followed the experimental protocols as described in [40].

To make use of the available labeled data, our model treats each sub-phrase as an independent sentence, and we learn the representation for all of the sub-phrases in the training set.

We initialized the word vectors with the unsupervised learning of word-level embedding using the word2vec algorithm, which implements continuous bag-of-words and skip-gram architectures for computing vector representations of a word.

The (Positive, Negative) presents results for the binary classification of sentences, and the fine-grained analysis predicts results for the case where five sentiment classes are used (positive, very positive, negative, very negative, and neutral). We report the accuracy of different methods in Table 4.3.

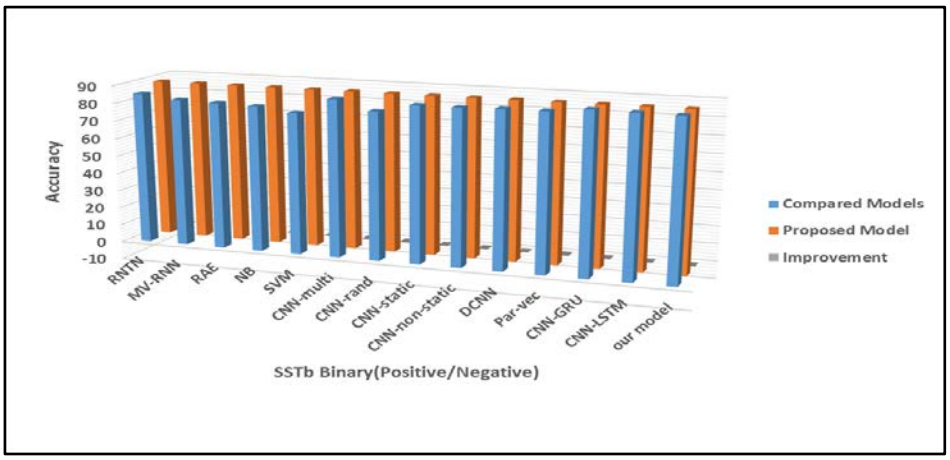

Figure 4.2. Accuracy on SSTb dataset for binary predictions

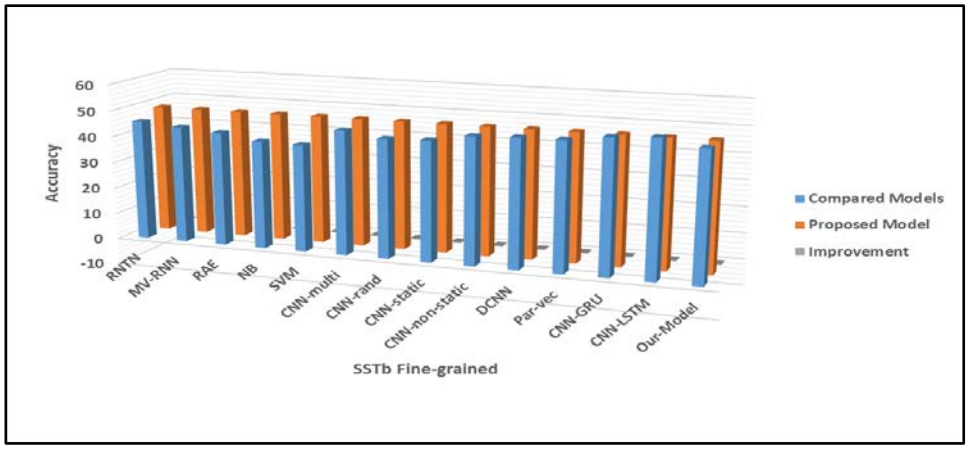

Figure 4.3. Accuracy on SSTb dataset for fine-grained (5-classes)

The primary highlight of our result on the SSTb benchmark dataset is that traditional methods (SVV, NB, BiNB) with bag-of-words perform poorly compared to our proposed deep learning language model. We observed 4%-12% absolute improvement in terms of accuracy with the baseline methods proposed in [15]. Initializing word-embeddings using unsupervised, pre-trained vectors gives the model an absolute accuracy that increased around 8% when compared to randomly initializing the vector with a CNN-only architecture [17]. Our model does not require pooling

layers, which leads to the more efficient capture of local information compared to the networks proposed in [6, 29]. The best previous result was reported by [40, 45] for SSTb. Our approach provides a 4% improvement in accuracy over the RNTN method. We also reported an 8% performance enhancement over the matrix-vector-RNN.

Parameter	**Fine-Grained**	**Binary**
RNTN [40]	45.7%	85.4%
MV-RNN [45]	44.4%	82.9%
RAE [80]	43.2%	82.4%
NB [40]	43.2%	82.4%
SVM [40]	41.0%	79.4%
CNN-Multi-channel [17]	47.1%	88.1%
CNN-rand [17]	45.0%	82.7%
CNN-static [17]	45.5%	86.8%
CNN-non-static [17]	48.0%	87.2%
DCNN [43]	48.5%	87.8%
Paragraph-Vec [24]	48.7%	87.8%
CNN-GRU-word2vec [12]	50.6%	89.9%
CNN-LSTM-word2vec [12]	51.5%	89.5%
Our approach	48.8%	89.2%

Table 4.3. The Performance of our approach compared to other approaches on SSTb dataset. The accuracy of fine-grained and binary predications are reported in the Table.

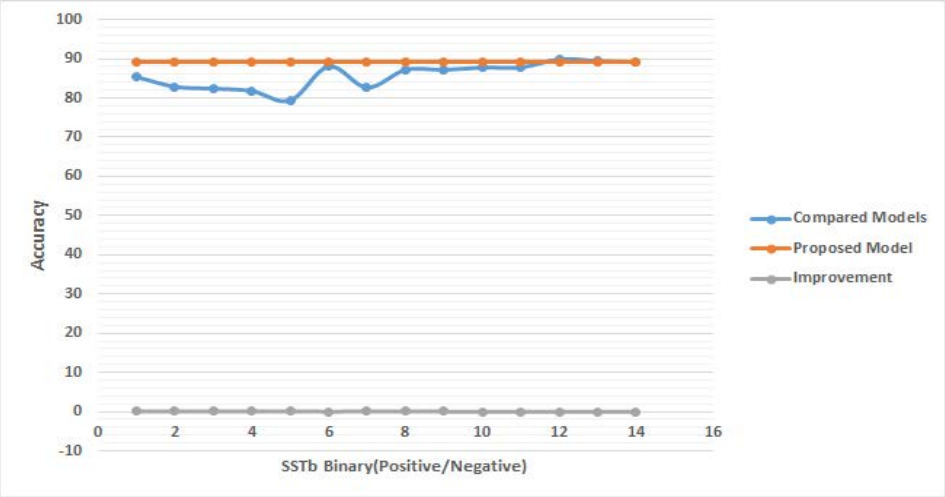

Figure 4.4. Predictions of positive and negative on SSTb.

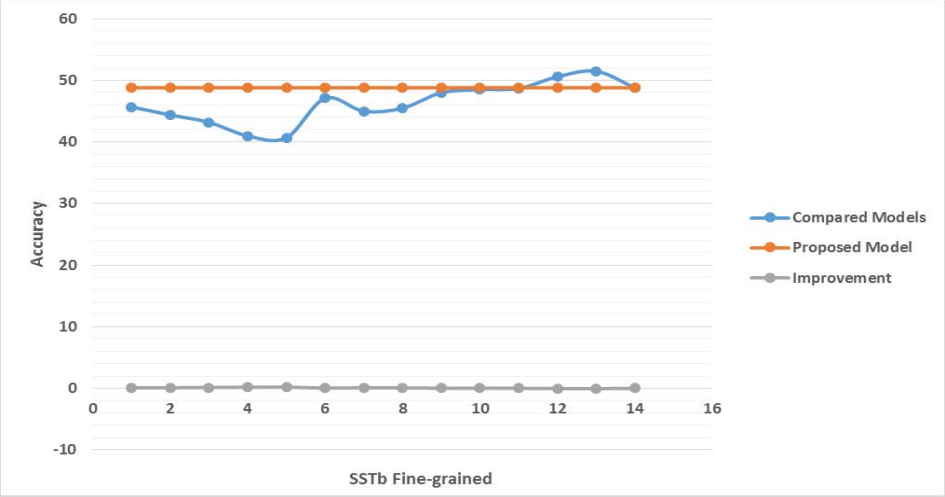

Figure 4.5. Prediction of fine-grained on SSTb.

In fine-grained classification tasks, our method has an absolute improvement of 7% in terms of accuracy. Figures 4.1 and 4.2 show that SSTb (binary and fine-grained), bag-of-n-words model, and (NB, SVM, BiNB) perform poorly on the dataset.

A similar model was proposed in [12] and achieved better performance in terms of accuracy; however these models have more hyperparameter and require subsampling

layers. On the other hand, our proposed model performed very competitively and came close to matching other state-of-the-art algorithms on both the binary and fine-grained sentiment analyses on the SSTb dataset with fewer parameters.

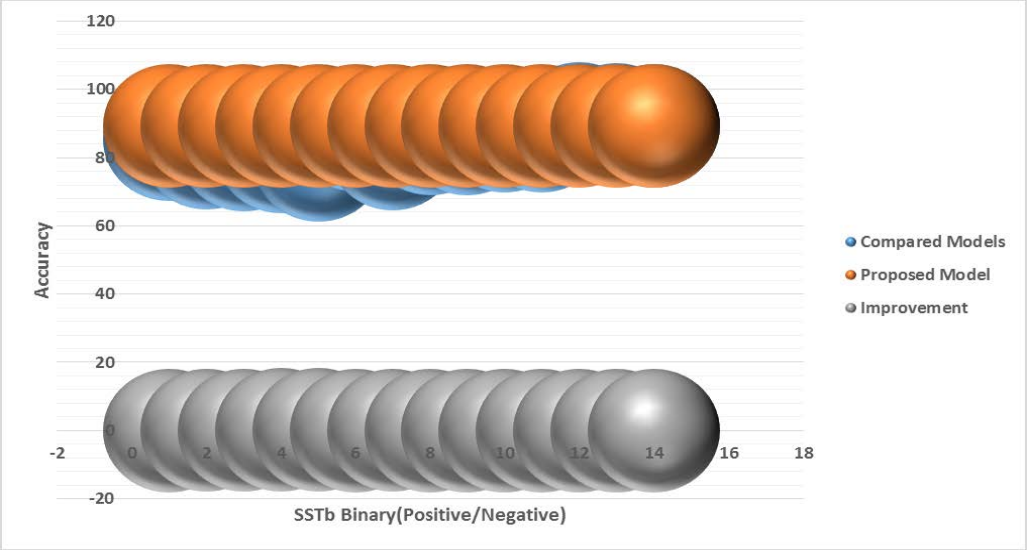

Figure 4.6. Results on SSTb dataset for binary predictions

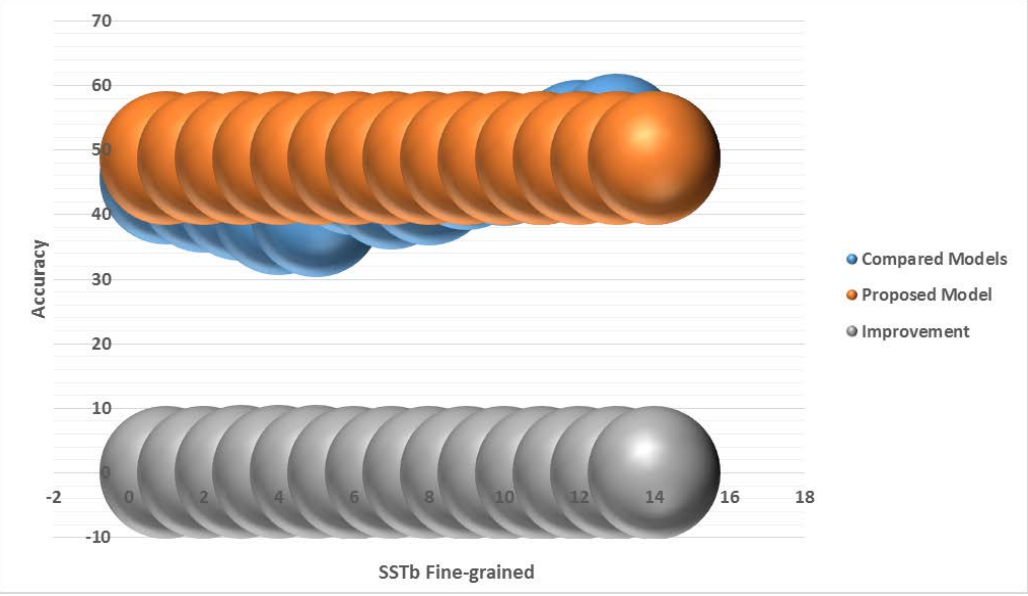

Figure 4.7. Results on SSTb dataset for fine-grained (5-classes)

4.3.2 ANALYSIS OF STANFORD LARGE MOVIE REVIEW DATASET (IMDB)

Beyond one sentence, each movie review consists of several sentences in the IMDB dataset. The results of our method are reported in Table 4.4 on the IMDB benchmark dataset compared to other approaches.

[40] Applied several methods on the IMDB dataset and found that their Recursive Neural Tensor Network worked much better than a bag-of-words model; however, this model required parsing and took into account the compositionality.

Our method performs better than all of the baselines reported in [15]: MNB-uni, MNB-bi, SVM-uni, SVM-bi, NBSVM-uni, and NBSVM-bi, with an approximate improvement of 2-12% in terms of accuracy.

When we compared the proposed model with a combined Restricted Boltzmann Machines model [106], bag-of-words, and WRRMB+ BoW (bnc), we achieved 4%-7% relative improvement and 1%-6% compared with Bow (bnc), Full+ Unlabeled + BoW, and paragraph vector [24].

The paragraph vectors proposed in [107] achieved a state-of-the-art result on the IMDB dataset; however the model has a reputation for being extremely difficult to tune and requires a downsampling parameter to reduce the feature map dimensionality for computational efficiency. We found that our proposed architecture, with no downsampling layer, achieved competitive results on the IMDB dataset as shown in Figure 4.3.

Parameter	Binary
MNB-uni [15]	83.5%
MNB-bi [15]	86.6%
SVM-uni [15]	86.9%
SVM-bi [15]	89.2%
NBSVM-uni [15]	88.3%
NBSVM-bi [15]	91.2%
WEEBM +Bow [106]	87.8%
WRBB + Bow (bnc) [106]	89.2%
BoW (bnc) [104]	87.8%
Full +Bow [104]	88.3%
Full +Unlabeled +Bow [104]	88.9%
Paragraph Vector [24]	92.5%
Paragraph Vector (LogReg) [23]	94.4%
Paragraph Vector (2-Layer MLP) [23]	94.5%
Our approach	93.2%

Table 4.4. The performance of our approach compared to other approaches on IMDB dataset. The accuracy of binary prediction.

The CNN-RNN with max-pooling loses detailed, local features due to the pooling layers in the architecture.

Compared with the existing methods and experiment results, we found that the approach takes advantage of both CNN and RNN models on the sentiment classification of short texts.

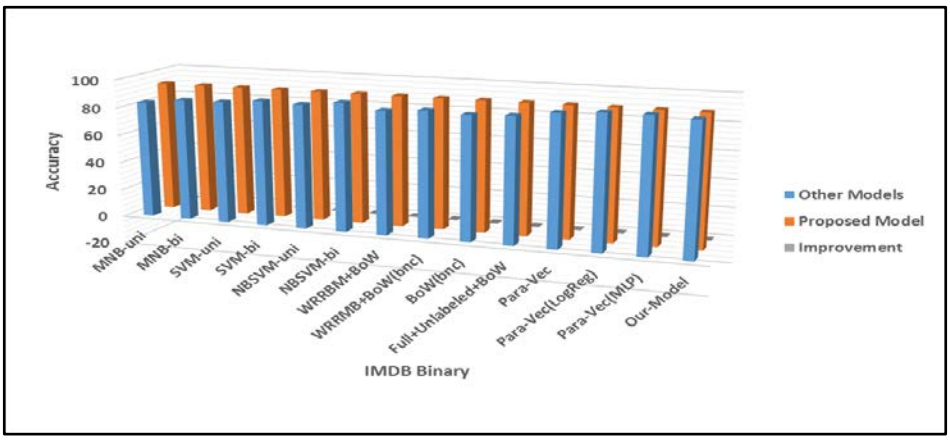

Figure 4.8. Accuracy for 2-classes on IMDB.

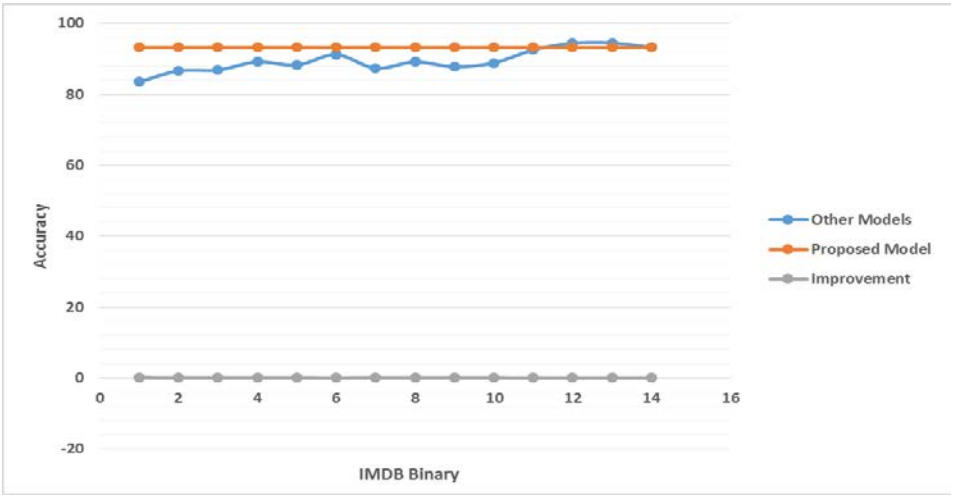

Figure 4.9. Prediction of positive and negative on IMDB.

Our experimental results suggest that by using a LSTM layer on top of a CNN architecture, one can effectively reduce the number of convolutional layers needed in order to capture long-term dependencies. Furthermore, we observed that many factors affect the performance of deep learning models, such as: the dataset size, vanishing and exploding of the gradients, and choosing the best feature extractors and classifiers, which are all still open research areas. However, there is no specific model for all types of datasets.

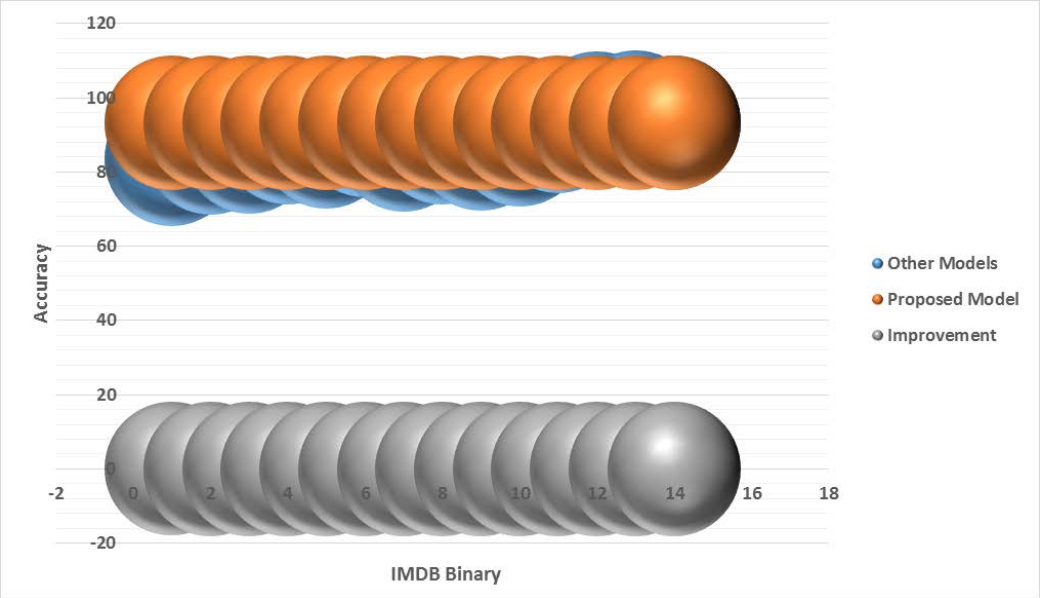

Figure 4.10. Results on IMDB dataset for binary predictions.

4.4 OVERVIEW

The challenge in NLP is to develop an architecture that can learn the hierarchal representation of the whole sentence jointly with the task. Convolutional neural networks consider feature extraction and classification as one jointly trained task.

The idea of CNNs has been improved upon recently [4, 5, 17, 29, 30] by using multiple layers of convolutional and pooling to sequentially extract hierarchal representation of input. Reducing the network size has been the interest of several works. More compact layers are also used, likely by replacing the fully connected layers with average pooling [49].

In [4] the weights are constrained by binary, which considerably reduces the memory consumption. To design a simpler network, [49] removed redundant

connections and allowed weight sharing. In our work we conducted a series of experiments with both deep learning and traditional methods to offer a fair comparison to competitive models on sentiment analysis benchmark datasets. Figure 4.4 show the proposed model compared to similar model.

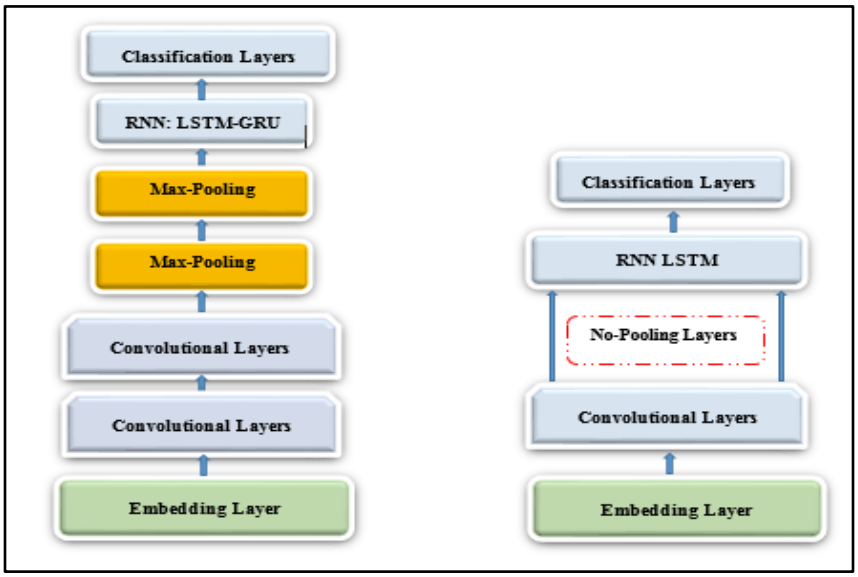

Figure 4.11. The proposed CNN-LSTM architecture compare to traditional CNN-RNN with max-pooling architecture.

We did our best to select the architectures that would deliver comparable and competitive results. Despite the fact that the CNN-RNN proposed in [12] has a slightly higher classification accuracy compared to our proposed model.

We argue that this result is due to the use of max pooling on adjacent words. However, our proposed architecture is simple and efficient in term of layers. Moreover, our model has significantly fewer parameters, which means less memory consumption.

We reported very competitive results in terms of accuracy in comparison to the model proposed in [12]. The reported result shows that, compared to the currently most

popular LSTM, CNN, and CNN-LSTM methods, our proposed framework can achieve similar or even better performance on sentiment analysis tasks.

CHAPTER 5: CONCLUSTION

Convolutional neural networks (CNN) learn to extract higher-level features that are invariant to local translation. Despite this advantage, it requires many layers of convolution to capture long-term dependencies, due to the locality of the convolutional and pooling. This becomes more severe as the length of the input sequence grows. Ultimately, this leads to the need for a very deep network with many convolutional layers. In this dissertation, we presented a new framework to overcome this problem. In particular, we aimed to capture the sub-word information and reduce the number of the parameters in the architecture.

Our framework jointly combines CNN and recurrent neural networks (RNN) on top of unsupervised, pre-trained word vectors; recurrent layers are expected to preserve ordering information even with one single layer. Thus, we exploited a recurrent layer as a substitute for the pooling layer to hypothetically reduce the loss of details in local information and capture long-term dependencies more efficiently. Our approach performed well on two benchmark datasets and achieved a competitive classification accuracy while outperforming several other methods. Our results demonstrated that it is possible to use a much smaller architecture to achieve the same level of classification performance.

REFERENCES

[1] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*, Berlin, Germany, pp. 137-142, 1998.

[2] Y. Mu, Y. Fan, L. Mao, and S. Han, "Event-related theta and alpha oscillations mediate empathy for pain," *Brain research,* vol. 1234, pp. 128-136, 2008.

[3] R. Collobert, "Deep Learning for Efficient Discriminative Parsing," in *AISTATS*, Fort Lauderdale, FL, USA, vol. 15, pp. 224-232, 2011.

[4] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, Helsinki, Finland, pp. 160-167, 2008.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research,* vol. 12, pp. 2493-2537, 2011.

[6] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very Deep Convolutional Networks for Natural Language Processing," *arXiv preprint arXiv:1606.01781,* 2016.

[7] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems*, Montreal, Quebec, Canada, pp. 3079-3087, 2015.

[8] C. N. Dos Santos and M. Gatti, "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts," in *In Proceedings of COLING, the 25th International Conference on Computational Linguistics, pp. 69-78.*, Dublin, Ireland, pp. 69-78, 2014.

[9] C. N. dos Santos and B. Zadrozny, "Learning Character-level Representations for Part-of-Speech Tagging," in *The 31st International Conference on Machine Learning*, Beijing, China, pp. 1818-1826, 2014.

[10] A. Hassan and A. Mahmood, "Convolutional Recurrent Deep Learning Model for Sentence Classification," *IEEE Access*, vol. 6, pp. 13949-13957, 2018.

[11] K. Vo, D. Pham, M. Nguyen, T. Mai, and T. Quan, "Combination of Domain Knowledge and Deep Learning for Sentiment Analysis," in *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, Gadong, Brunei Darussalam, pp. 162-173, 2017.

[12] X. Wang, W. Jiang, and Z. Luo, "Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts," in *The 26th International Conference on Computational Linguistic*, Osaka, Japan, pp. 2428-2437, 2016.

[13] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*, pp. 137-186, 2006.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781,* 2013.

[15] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, Jeju Island, Korea, vol 2, pp. 90-94, 2012.

[16] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic, 2007.

[17] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882,* 2014.

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473,* 2014.

[19] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence,* vol. 35, no. 8, pp. 1798-1828, 2013.

[20] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *journal of machine learning research*, vol. 3, pp. 1137-1155, 2003.

[21] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850,* 2013.

[22] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *International Conference on Acoustics, Speech and Signal Processing*, Vancouver, Canada, pp. 6645-6649, 2013

[23] J. Hong and M. Fang, "Sentiment analysis with deeply learned distributed representations of variable length texts," Technical report, Stanford University, CA, USA, pp. 655-665, 2015.

[24] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *The 31st International Conference on Machine Learning*, Beijing, China, vol. 14, pp. 1188-1196, 2014.

[25] G. LU, "Word representations: a simple and general method for semi-supervised learning," in Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, pp. 384-394, 2015.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, Lake Tahoe, Nevada, USA, pp. 3111-3119, 2013.

[27] T. N. Sainath *et al.*, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39-48, 2015.

[28] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning*, Bellevue, Washington, USA, pp. 129-136, 2011.

[29] Y. Xiao and K. Cho, "Efficient Character-level Document Classification by Combining Convolution and Recurrent Layers," *arXiv preprint arXiv:1602.00367*, 2016.

[30] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, Montreal, Quebec, Canada, pp. 649-657, 2015.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature 323*, p. 533-536, 1986.

[32] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed representations," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, no. 3, pp. 77-109, 1986.

[33] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for training large scale neural network language models," in *Automatic Speech Recognition and Understanding*, Hawaii, USA, pp. 196-201, 2011.

[34] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černocký, "Empirical evaluation and combination of advanced language modeling techniques," in *Twelfth Annual Conference of the International Speech Communication Association*, Florence, Italy, 2011.

[35] H. Schwenk, "Continuous space language models," *Computer Speech & Language*, vol. 21, no. 3, pp. 492-518, 2007.

[36] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," *arXiv preprint arXiv:1607.01759*, 2016.

[37] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "Learning semantic representations using convolutional neural networks for web search," in *Proceedings of the 23rd International Conference on World Wide Web*, Seoul, Republic of Korea, pp. 373-374, 2014.

[38] Yih, W.T. He, X, and Meek, C, "Semantic Parsing for Single-Relation Question Answering," in *in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics,* Baltimore, Maryland, USA, vol2, pp. 643-648, 2014.

[39] J. Weston, S. Chopra, and K. Adams, "# TagSpace: Semantic embeddings from hashtags," presented at the Proceedings of the Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, pp. 1822-1827, 2014.

[40] R. Socher *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the conference on empirical methods in natural language processing*, Seattle, Washington, USA, pp. 1631-1642, 2013.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, Lake Tahoe, CA, USA, pp. 1097-1105, 2012.

[42] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058*, 2014.

[43] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[44] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 23, no. 3, pp. 517-529, 2015.

[45] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng, "Semantic compositionality through recursive matrix-vector spaces," in *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Jeju Island, Korea, pp. 1201-1211, 2012.

[46] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391-407, 1990.

[47] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and trends in information retrieval*, vol. 2, no. 1-2, pp. 1-135, 2008.

[48] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[49] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, Madison, Wisconsin, USA, vol. 752, pp. 41-48, 1998.

[50] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359-394, 1999.

[51] A. Mnih and G. Hinton, "Three new graphical models for statistical language modelling," in *Proceedings of the 24th international conference on Machine learning*, Corvallis, OR, USA, pp. 641-648, 2007.

[52] P. F. Brown, V. J. D. Pietra, R. L. Mercer, S. A. D. Pietra, and J. C. Lai, "An estimate of an upper bound for the entropy of English," *Computational Linguistics*, vol. 18, no. 1, pp. 31-40, 1992.

[53] H. Schütze, "Distributional part-of-speech tagging," in *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, Dublin, Ireland, pp. 141-148, 1995.

[54] S. Miller, J. Guinness, and A. Zamanian, "Name tagging with word clusters and discriminative training," in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, Boston, MA, USA, 2004.

[55] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, Boulder, Colorado, USA, pp. 147-155, 2009.

[56] T. Koo, X. Carreras, and M. Collins, "Simple semi-supervised dependency parsing," in *Proceedings of ACL*, Columbus, Ohio, USA, pp. 595-603, 2008.

[57] D. Lin and X. Wu, "Phrase clustering for discriminative learning," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Singapore, pp. 1030-1038, 2009.

[58] F. Huang and A. Yates, "Distributional representations for handling sparsity in supervised sequence-labeling," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Singapore, pp. 495-503, 2009.

[59] R. Socher, "Recursive deep learning for natural language processing and computer vision," , Stanford University, CA, USA, 2014.

[60] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," ed: A field guide to dynamical recurrent neural networks, 2001.

[61] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179-211, 1990.

[62] H. Schwenk and J.-L. Gauvain, "Training neural network language models on very large corpora," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Vancouver, British Columbia, Canada, pp. 201-208, 2005.

[63] A. Emami and F. Jelinek, "Exact training of a neural syntactic language model," in *Acoustics, Speech, and Signal Processing*, Monreal, Quebec, Canada, vol. 1, pp. I-245, 2004.

[64] A. Alexandrescu and K. Kirchhoff, "Factored neural language models," in *Proceedings of the Human Language Technology Conference of the NAACL*, New York, NY, USA, pp. 1-4, 2006.

[65] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, Makuhari, Japan, 2010.

[66] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *International Conference on Acoustics, Speech and Signal Processing*, Prague, Czech Republic, pp. 5528-5531, 2011.

[67] J. Goodman, "Classes for fast maximum entropy training," in International Conference of *Acoustics, Speech, and Signal Processing*, Salt Lake, Utah, USA, pp. 561-564, 2001.

[68] T. Mikolov, J. Kopecky, L. Burget, and O. Glembek, "Neural network based language models for highly inflective languages," in *International Conference on Acoustics, Speech and Signal Processing*, Taipei, Taiwan, pp. 4725-4728, 2009.

[69] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," *arXiv preprint arXiv:1702.00887,* 2017.

[70] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations," in *The Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Atlanta, GA, USA, pp. 746-751, 2013.

[71] L. C. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille, "Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform," *arXiv preprint arXiv:1511.03328,* 2015.

[72] J. Gao, L. Deng, M. Gamon, X. He, and P. Pantel, "Modeling interestingness with deep neural networks," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, pp. 2-13, 2014.

[73] A. Hassan and A. Mahmood, "Deep Learning approach for sentiment analysis of short texts," in *The 3rd International Conference on Control, Automation and Robotics*, Nagoya, Japan, pp. 705-710, 2017

[74] A. Hassan and A. Mahmood, "Deep learning for sentence classification," presented at the Systems, Applications and Technology Conference, Long Island, NY, USA, 2017.

[75] S. Albelwi And A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks." Entropy, vol. 19, no 9, p. 242, 2017.

[76] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," *arXiv preprint arXiv:1508.06615,* 2015.

[77] A. G. Ororbia II, T. Mikolov, and D. Reitter, "Learning simpler language models with the differential state framework," *Neural computation,* vol. 29, no. 12, pp. 3327-3352, 2017.

[78] J. B. Pollack, "Recursive distributed representations," *Artificial Intelligence,* vol. 46, no. 1-2, pp. 77-105, 1990.

[79] A. Küchler and C. Goller, "Inductive learning in symbolic domains using structure-driven recurrent neural networks," in *Annual Conference on Artificial Intelligence*, pp. 183-197, 1996.

[80] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the conference on empirical methods in natural language processing*, Edinburgh, United Kingdom, pp. 151-161, 2011.

[81] K. M. Hermann and P. Blunsom, "The role of syntax in vector space models of compositional semantics," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, Sofia, Bulgaria, vol. 1, pp. 894-904, 2013.

[82] F. A. Gers and E. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks,* vol. 12, no. 6, pp. 1333-1340, 2001.

[83] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, pp. 1700-1709, 2013.

[84] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, Montreal, Quebec, Canada, pp. 577-585, 2015.

[85] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[86] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[87] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," in *International Conference on Machine Learning*, Beijing, China, pp. 584-592, 2014.

[88] C. Szegedy *et al.*, "Going deeper with convolutions," presented at the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1-9, Boston, MA, USA, 2015.

[89] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[90] A. Hassan and A. Mahmood, "Efficient Deep Learning Model for Text Classification Based on Recurrent and Convolutional Layers," presented at the The 16th International Conference on Machine Learning and Applications, Cancun, Mexico, 2017.

[91] N. M. Mayer, "Echo State Condition at the Critical Point," *Entropy*, vol. 19, no. 1, p. 3, 2016.

[92] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks,* vol. 5, no. 2, pp. 157-166, 1994.

[93] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," presented at the The 30th International Conference on Machine Learning, Atlanta, GA, USA, pp. 1310-1318, 2013.

[94] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural computation,* vol. 12, no. 10, pp. 2451-2471, 2000.

[95] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," in *International Joint Conference on Neural Networks*, Montreal, QC, Canada, vol. 18, no. 5, pp. 602-610, 2005.

[96] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *arXiv preprint arXiv:1503.04069,* 2015.

[97] T. Mikolov, "Language Modeling for Speech Recognition in Czech," Masters thesis, Brno University of Technology, Brno, Czechia, 2007.

[98] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing,* vol. 45, no. 11, pp. 2673-2681, 1997.

[99] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics,* vol. 15, no. 11, pp. 937-946, 1999.

[100] T. K. Landauer, Foltz, P.W., and Laham, D, " *An introduction to latent semantic analysis,"* Discourse processes, vol 25, no 2-3, pp. 259-284, 1998.

[101] M. Baroni, G. Dinu, and G. Kruszewski, "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors," in *The 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, Maryland, USA, vol. 1, pp. 238-247, 2014.

[102] M. Boden, "A guide to recurrent neural networks and backpropagation," *The Dallas project, SICS technical report,* 2002.

[103] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on association for computational linguistics*, Ann Arbor, Michigan, pp. 115-124, 2005.

[104] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, Oregon, pp. 142-150, 2011.

[105] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580,* 2012.

[106] G. E. Dahl, R. P. Adams, and H. Larochelle, "Training restricted boltzmann machines on word observations," *arXiv preprint arXiv:1202.5695,* 2012.

[107] J. Hong and M. Fang, "Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts," Technical report, Stanford University, CA, USA, pp. 655-665, 2015.